VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Jakub Jedlička

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

ŠIFRÁTOR PRO HARDWAROVĚ OMEZENÉ ZAŘÍZENÍ
HARDWARE FOR LIGHTWEIGHT CRYPTOGRAPHIC IMPLEMENTATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jakub Jedlička**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. David Smékal**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Jakub Jedlička ***ID:*** 198597

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Šifrátor pro hardwarově omezené zařízení

POKYNY PRO VYPRACOVÁNÍ:

V rámci bakalářské práce se student seznámí s hardwarovou akcelerací založenou na platformě FPGA. Seznamte se s programovacím jazykem VHDL, FPGA obvody řady Zynq-7000 a platformou Xilinx Vivado. Nastudujte šifrovací algoritmy určené pro lehkou kryptografii. Následně šifru popište jazykem VHDL, proveďte simulaci a implementujte návrh na hardwarové zařízení FPGA s omezenými hardwarovými zdroji.
Věnujte pozornost konkrétnímu FPGA obvodu s čipem Artix s omezenými hardwarovými zdroji a zprovozněte šifrátor na vývojové desce ZYBO Z7. Změřte dosažené výsledky a porovnejte je s jiným existujícím řešením.
Úkolem je také zprovoznění jednotlivých periferií komunikující s FPGA obvodem Artix (komunikace pomocí rozhraní USB 2.0, Micro SD, Ethernet, a jiné).

DOPORUČENÁ LITERATURA:

[1] BURDA, Karel. Aplikovaná kryptografie. Brno: VUTIUM, 2013. ISBN 978-80-214-4612-0.

[2] PINKER, Jiří, Martin POUPA. Číslicové systémy a jazyk VHDL. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-7300-198-5.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Ing. David Smékal

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

UPOZORNĚNÍ:
Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Tato bakalářská práce se věnuje tématu lehké kryptografie a implementaci vybrané šifry na programovatelné hradlové pole (FPGA). Práce se prvně zabývá teorií, kde jsou popsány hardwarové prvky, obecná kryptografie a lehká kryptografie se zaměřením na šifry LBlock a PRESENT. Následně popisuje výběr typu šifry a poté výběr konkrétní šifry lehké kryptografie. Dále je vybrána šifra LBlock, implementována a otestována v podobě vlastního „Intelectual Property“ (IP) bloku pomocí hardwarového deskriptivního jazyka pro velmi rychlé integrované obvody (VHDL). Tento blok je využit v blokovém designu, pomocí kterého je realizován šifrátor na vývojové desce ZYBO-Z7. Práce se vstupními a výstupní daty je realizována na procesním systému čipu Zynq-7000, který data předává programovací logice. V závěru je tato komunikace a implementace popsána, kdy jsou pro šifru LBlock použity operační módy – šifrové zpětné vazby a výstupní zpětné vazby. Pro tyto operační módy jsou provedena měření ke zjištění rychlostí šifrování dat uložených na microSD kartě a popsány úskalí vyplývající při tomto šifrování.

KLÍČOVÁ SLOVA

Šifrátor, ZYBO-Z7, FPGA, Zynq-7000, IP blok, blokové šifry, lehká kryptografie, LBlock, PRESENT, CFB, OFB

ABSTRACT

This bachelor thesis deals with the topic of lightweight cryptography and the implementation of a selected cipher on a field programmable gate array (FPGA). Thesis first deals with the theory, where hardware elements, general cryptography and lightweight cryptography are described with focus on the LBlock and PRESENT ciphers. It then describes the selection of a cipher type and then the selection of a particular lightweight cryptography cipher. Next the LBlock cipher is selected, implemented, and tested as a custom intellectual property (IP) block using a hardware descriptive language for very fast integrated circuits (VHDL). This block is used in the block design to implement the encryptor on the ZYBO-Z7 development board. The input and output data handling is implemented on the Zynq-7000 processing system chip, which passes the data to the programmable logic. Finally this communication and implementation is described where the operational modes used for the LBlock cipher are cipher feedback mode and output feedback mode. For these operational modes, measurements are made to determine the encryption speed of the data stored on the microSD card and the pitfalls resulting from this encryption are described.

KEYWORDS

Encryptor, ZYBO-Z7, FPGA, Zynq-7000, IP block, block ciphers, lightweight cryptography, LBlock, PRESENT, CFB, OFB

Vysázeno pomocí balíčku `thesis` verze 4.03; `http://latex.feec.vutbr.cz`

JEDLIČKA, Jakub. *Hardwarový šifrátor pro lehkou kryptografii*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 81 s. Bakalářská práce. Vedoucí práce: Ing. David Smékal

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Jakub Jedlička
VUT ID autora:	198597
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Hardwarový šifrátor pro lehkou kryptografii

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Davidu Smékalovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat rodině za trpělivost, podporu při studiu a vytváření této práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

V dnešní době stále rostou požadavky nejen na rychlost, ale i bezpečnost. Proto je jedním z vhodných řešení využití čipů FPGA (Programovatelné hradlové pole – Field-Programmable Gate Array), které jsou škálovatelné s aktuálními potřebami. Čipy FPGA mají výhodu oproti CPU (Central Processing Unit) v jejich rychlosti zpracování instrukcí, které je mnohonásobně rychlejší. V současnosti se FPGA používá společně s CPU, kdy FPGA se stará o rychlost provedení jednoduchých procesů. Využitím FPGA čipů společně s šiframi lehké kryptografie lze docílit šifrování značně rychlejšího než za použití samostatného CPU a šifer kryptografie, které jsou hardwarově náročnější a tím jejich doba šifrování trvá déle za cenu snížení bezpečnosti.

Proto se tato bakalářská práce věnuje šifrám lehké kryptografie a následnému výběru a implementaci vhodně zvolené šifry pro vytvoření šifrátoru, ke kterému slouží vývojová deska ZYBO Z7, osazena FPGA čipem řady Zynq-7000. Práce je rozdělena do 10 kapitol, kde první tři obsahují teoretickou část a dalších sedm obsahuje praktickou část.

V první kapitole jsou blíže přiblíženy FPGA obvody, FPGA čip řady Zynq-7000 a vývojová deska ZYBO Z7. Prvně jsou v ní obecně popsány jednotlivé části a jejich rozložení na FPGA čipu. Dále je detailněji představen a popsán FPGA čip z řady Zynq-7000 od jednoho z největších výrobců firmy Xilinx, Inc. Nakonec je popsána vývojová deska ZYBO Z7, která bude sloužit jako šifrátor, jelikož obsahuje rozdílné typy vstupů pro data.

V druhé kapitole jsou objasněny základy kryptografie, v jejíž první části jsou definovány základní pojmy používané v celé práci a v navazující části jsou popsány hašovací funkce, asymetrické a symetrické kryptosystémy. Větší pozornost je věnována symetrickým kryptosystémům a jejich provozním režimům.

Ve třetí kapitole je detailně rozebrán princip lehké kryptografie a její využití na FPGA. Zde jsou obecně analyzovány kryptosystémy a jejich známí zástupci, ze kterých jsou podrobněji popsány dvě šifry zastupující blokové šifry lehké kryptografie.

Ve čtvrté kapitole je popsán postup výběru algoritmu, který bude následně implementován na FPGA čip. V první části jsou analyzovány výhody a nevýhody implementace daných typů kryptosystémů pro šifrování zdrojových dat ve formě souborů. V další jsou porovnány konkrétní algoritmy pro implementaci na FPGA čipy z pohledu bezpečnosti a výkonnosti. Z těchto algoritmů je nakonec vybrána konkrétní šifra lehké kryptografie.

V páté kapitole jsou popsány dvě využitá vývojová prostředí, která byla použita při vývoji šifrátoru.

V šesté kapitole je rozebrána implementace vybrané šifry lehké kryptografie do hardwarově deskriptivního jazyka pro velmi rychlé integrované obvody (VHDL).

Tato implementace je následně testována a její výstupy jsou validovány. Nakonec je zde popsán posuvný registr s lineární zpětnou vazbou (LFSR).

V sedmé kapitole je vysvětlen pojem „Intelectual Property“ (IP) blok a poté popsána integrace vybrané a implementované šifry do vlastního IP bloku.

V osmé kapitole je představen navržený blokový návrh, který je použit pro šifrátor implementovaný na vývojové desce. Dále je popsána funkcionalita jednotlivých částí blokového návrhu. Nakonec jsou analyzovány hardwarové požadavky návrhu a jeho částí.

V deváté kapitole je prvně uveden popis komunikace procesního systému s jednotlivými prvky blokového návrhu. V další části je objasněn průběh šifrování šifrátoru z různých typů vstupů. Pro tyto typy šifrování jsou následně provedeny a zanalyzovány měření rychlosti šifrování.

V poslední desáté kapitole porovnáván vytvořený šifrátor s jinými implementacemi šifer LBlock, PRESENT a AES.

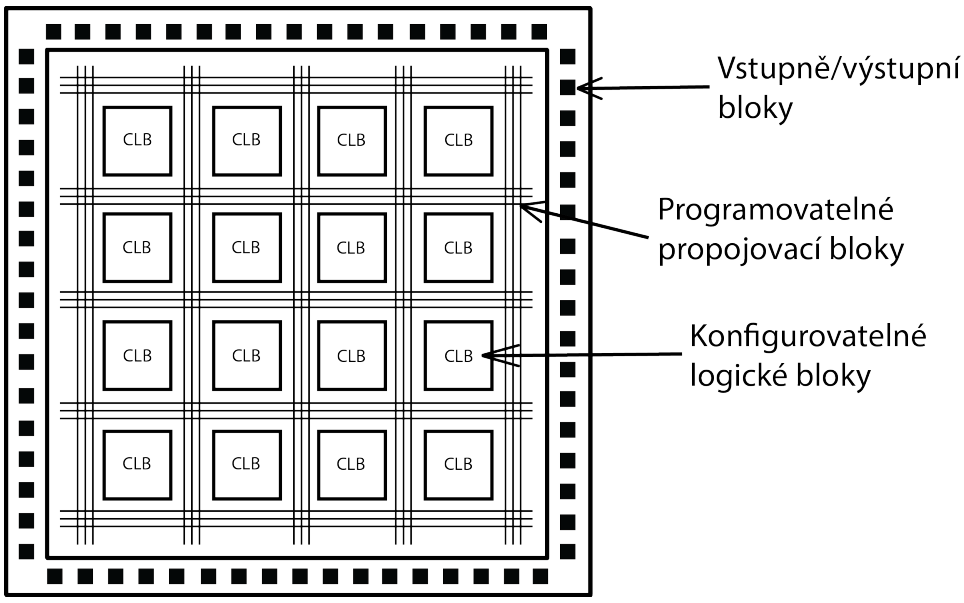
1 Programovatelné hradlové pole

První kapitola se věnuje obecnému popisu programovatelných hradlových polí neboli FPGA[1] a jejich částí. Následující části se zabývají konkrétním typem FPGA čipu z řady Zynq-7000 a vývojovou deskou ZYBO Z7, na které je tento FPGA čip implementován.

1.1 Architektura programovatelných hradlových polí

Programovatelná hradlová pole byla poprvé představena v 80. letech společností Xilinx, Inc. Do nedávna byly FPGA používány k prototypování a emulacím při procesu návrhu a výroby integrovaných obvodů určených pro aplikaci tzv. ASIC[2] [1]. Nicméně v poslední době se FPGA čipy rozšířily mimo návrh ASIC obvodů a jsou vyráběny v mnoha variantách pro různá odvětví. Hlavní výhodou obvodů FPGA je možnost následného přeprogramování i po jejich výrobě [2].

FPGA se skládají ze tří hlavních typů programovatelných stavebních bloků. Rozložení těchto tří bloků je zobrazeno na obrázku 1.1.



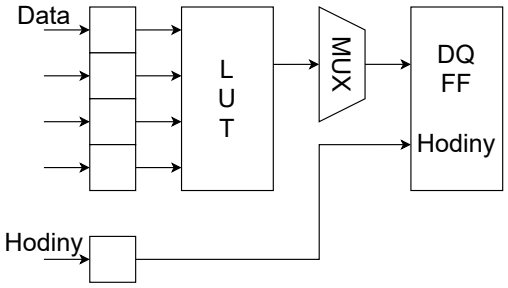
Obr. 1.1: Architektura FPGA čipu [3].

- IOB - vstupně/výstupní bloky
- PIB - programovatelné propojovací bloky
- CLB - konfigurovatelné logické bloky

[1] Programovatelná hradlová pole z anglického Field-Programmable Gate Array

[2] Integrované obvody určené pro aplikaci z anglického Application Specific Integrated Circuit

CLB mohou být naprogramovány tak, aby implementovaly různé logické funkce. Tyto bloky se skládají z takzvaných LUT (Look-Up Table). Dále obsahují bistabilní klopné obvody (Flip-Flops) a multiplexory. Logické zapojení CLB je zobrazeno na obrázku 1.2. CLB prvky jsou vzájemně propojeny pomocí programovatelných propojovacích bloků a vytvářejí velké logické okruhy. Tyto logické okruhy jsou z vnější částí propojovány pomocí IOB. Na FPGA čipu se dále nachází SRAM[3], která uchovává konfiguraci CLB, PIB a IOB bloků. Společně s SRAM se musí na FPGA čipu nacházet další typ paměti, který je nezávislý na energii a obsahuje konfigurační soubor pro nastavení SRAM při odstávce energie [1][4][5].



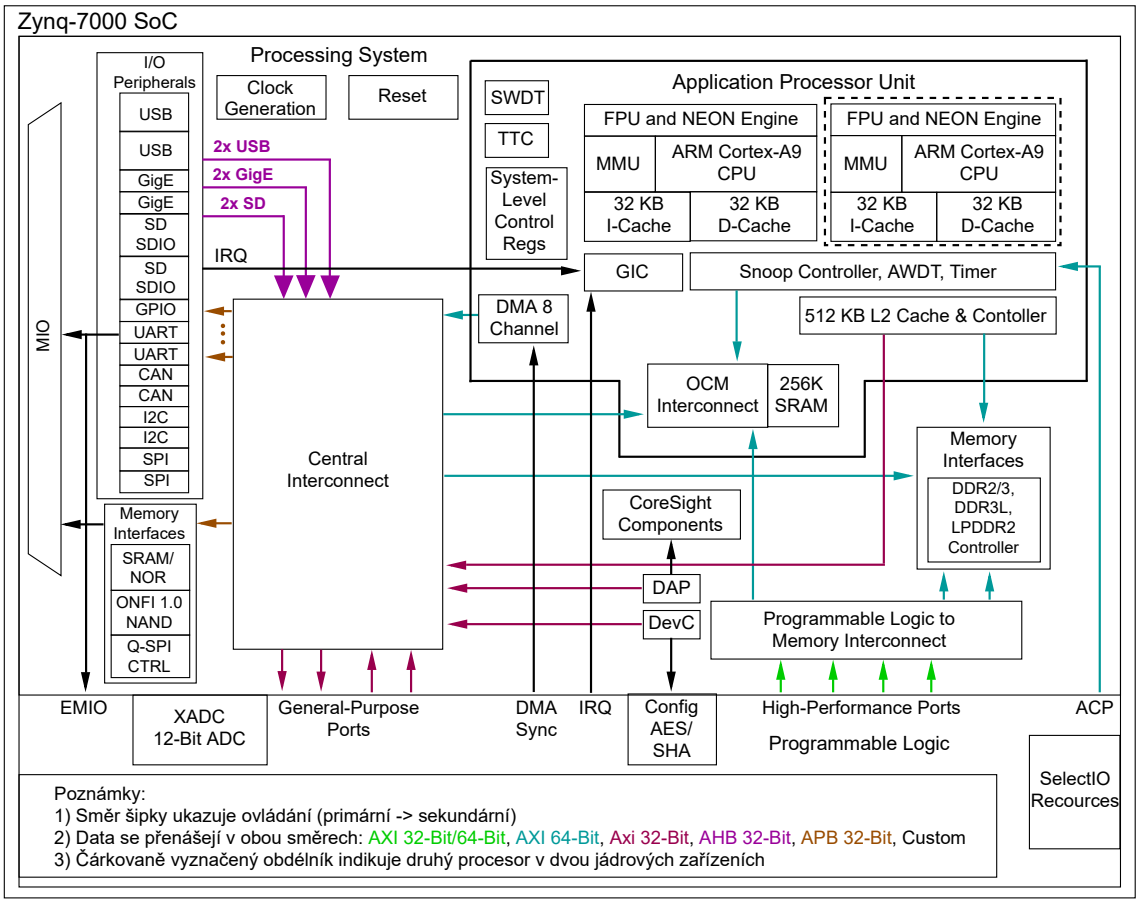
Obr. 1.2: Architektura logického bloku [5].

1.2 Xilinx Zynq-7000

Čipy z řady Zynq-7000, architektura zobrazena na obrázku 1.3, nabízí flexibilitu a škálovatelnost FPGA čipů, zatímco poskytují sílu, výkon a použitelnost, která je často spojována s ASIC čipy. O výpočetní sílů se stará procesní systém (PS), který je založen na technologii ARM©[4] Cortex™-A9, obsahující paměť přímo na čipu, rozhraní pro externí paměť a sadu periferních rozhraní. FPGA se často označuje jako programovatelná logika (PL), kde pro Zynq-7000 je založen na bázi 28nm výrobního procesu od firmy Xilinx. Výhodou řady Zynq-7000 je, že processing system a programmable logic jsou integrovány na jednom čipu. Touto integrací lze dosáhnout většího výkonu než když jsou tyto dvě části rozděleny na více čipech. Vývoj pro všechny typy programmable logic od Xilinxu probíhá v prostředí Vivado Design Suite [6].

[3] Statická paměť.

[4] Advanced RISC machine je typ procesoru, který má zmenšenou sadu instrukcí a je energeticky nenáročný.



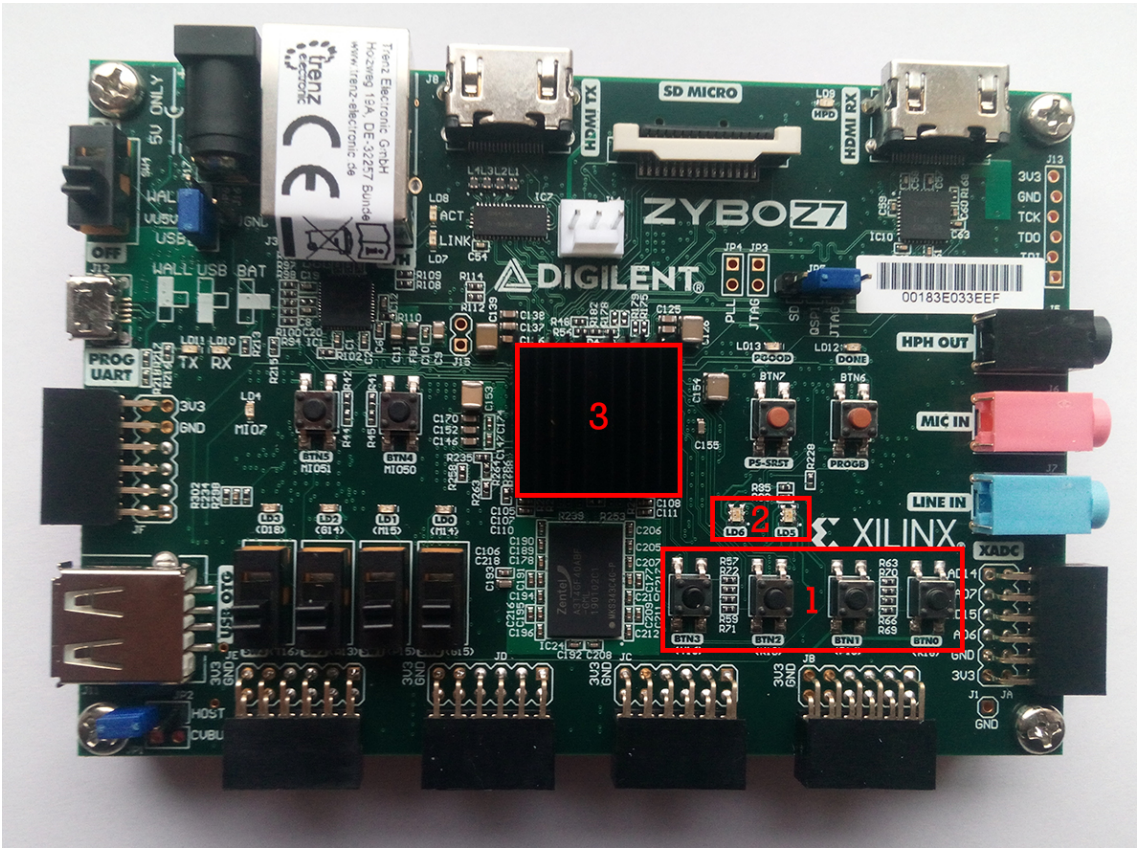
Obr. 1.3: Architektura Zynq-7000 čipu [6].

1.3 ZYBO Z7

Vývojová deska ZYBO Z7 je navržena a vyvíjena společností Digilent Inc. Jádrem vývojové desky je FPGA čip Zynq-7000, který je popsaný v kapitole 1.2 Xilinx Zynq-7000. FPGA čip je na desce obklopen sadou rozhraní pro připojení periferií, které slouží jak vstupní tak i výstupní. Obsahuje rozhraní pro microSD kartu a USB (Universal Serial Bus), které mají za úkol pracovat s daty. Pro komunikaci s deskou se využívá jedno ze dvou UART (Universal Asynchronous Receiver-Transmitter) rozhraní, přičemž druhé slouží k práci s daty. Pro síťový provoz se používá rozhraní RJ45. Deska dále obsahuje paměť typu DDR3L o velikosti 1 GiB. Pro zobrazení nebo zpracování grafického vstupu a výstupu jsou využita dvě HDMI (High-Definition Multimedia Interface) rozhraní, kdy jedno slouží jako vstupní a druhé jako výstupní. ARM procesor operuje na maximální frekvenci 667 MHz a řadič DDR3 paměti na maximální frekvenci 533 MHz [7].

Deska ZYBO Z7 je vyráběna ve dvou variantách, které se liší na základě parametrů. Na obrázku 1.4 je znázorněna její výkonnější varianta ZYBO Z7-20. Tato verze

obsahuje Zynq-7000 s označením XC7Z020–1CLG400C. Na čipu se nachází 53 200 vyhledávacích tabulek (LUT) a 106 400 bistabilních klopných obvodů (Flip-Flop). V této práci jsou používány 4 tlačítka a 2 RGB LED (Elektroluminiscenční dioda s červenou, zelenou a modrou barvou), které jsou součástí vývojové desky. Tlačítka a RGB LED komunikují přímo s PL a jsou na obrázku 1.4 označena číslem 1 a RGB LED číslem 2. Číslo 3 označuje čip ZYNQ, který je umístěn pod chladičem [7].



Obr. 1.4: Vývojová deska ZYBO Z7-20.

2 Kryptografie

V této kapitole jsou probrány základní principy a pojmy z kryptografie. První část je zaměřena na seznámení se základními pojmy kryptologie a principem kryptosystémů. Druhá část popisuje jednotlivé typy kryptosystémů dle jejich principu.

2.1 Základní pojmy

Kryptologie je věda zabývající se konstrukcí a překonáváním matematických metod zajišťující důvěrnost a autentičnost zpráv. Tato věda v sobě obsahuje kryptografii a kryptoanalýzu [8].
Kryptografie je věda zabývající se konstrukcí matematických metod pro zajištění důvěrnosti a autentičnosti zpráv [8].
Kryptoanalýza je věda zabývající se překonáváním matematických metod, které zajišťují důvěrnost a autentičnost zpráv [8].
Kryptografický systém zkráceně kryptosystém je systém algoritmů, který zajišťuje důvěrnost a autentičnost zpráv [8].
Klíč je parametr procesu šifrování a dešifrování [9].
Zpráva jsou vstupní data, která jsou potřeba šifrovat, někdy označována jako čistý nebo prostý text [10].
Šifrovaný text jsou výstupní data z procesu šifrování, která jsou v nečitelné podobě [8].
Šifra je určitý postup neboli algoritmus pomocí kterého se šifruje a dešifruje zpráva [9].
Šifrování je proces transformace určité zprávy do nečitelné podoby [10].
Šifrátor je zařízení, které šifruje zprávu pomocí šifry na šifrovaný text.
Dešifrování se nazývá opačný proces šifrování neboli přetváříme šifrovaný text na původní zprávu [10].

2.2 Princip kryptosystémů

V roce 1883 Auguste Kerchoffs sepsal podmínky pro vytváření kryptosystémů, kde jedna z podmínek se rozšířila a stala se uznávanou konvencí známou jako Kerchoffsův princip, který říká:

> *„Znalost algoritmu a velikosti klíče stejně jako dostupnost známého prostého textu jsou standardní předpoklad v moderní kryptoanalýze. Protože útočník může tyto informace případně získat, není proto dobré se spoléhat na jeho utajení při posuzovaní jeho kryptografické síly.*[1] “

[1] Přeloženo z anglického jazyka, originální znění v Modern Cryptography na straně 208 [10] .

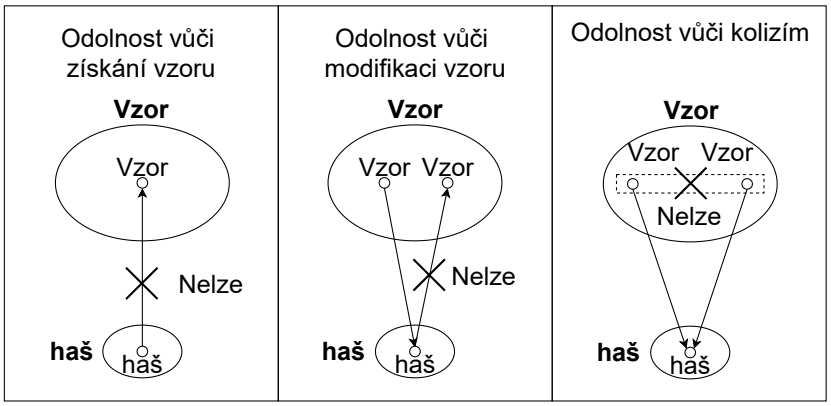
Propojením Kerchoffsova principu a Shannonovy teorie informace lze poskytnout shrnutí pro dobře navržený kryptografický systém:

- Algoritmus by neměl obsahovat žádnou část, která je tajná.
- Rozložení informace v zašifrovaném textu by mělo být po celé délce rovnoměrné až náhodné.
- Správný klíč by měl zaručit, aby šifrování a dešifrování bylo efektivní.
- Velikost klíče by měla udávat náročnost na dešifrování zašifrovaného textu bez použití správného klíče [10].

2.3 Hašovací funkce

Jednosměrné funkce jsou takové, pro které lze vypočítat obraz funkce, ale z obrazu by nemělo být možné vypočítat vzor obrazu. Jednosměrné funkce se dělí na funkce s volitelnou a pevnou délkou výstupu [8]. Skupina s pevnou délkou výstupu se nazývá hašovací funkce. Hašovací funkce pracuje na principu, kde vstup je bitový řetězec proměnné délky a výstupem je řetězec o konstantní bitové délce takzvaný haš[2] [10]. Od hašovacích funkcí jsou požadovány tyto vlastnosti (zobrazeny na obrázku 2.1):

- Odolnost vůči získaní vzoru
- Odolnost vůči modifikaci vzoru
- Odolnost vůči kolizím
- Efektivita



Obr. 2.1: Požadované vlastnosti haš funkce [8].

Jednosměrnost neboli odolnost vůči získání vzoru by měla splňovat, že z množiny hašů by nemělo být prakticky možné nalézt vzor, pro který by platila funkce H(vzor) = haš, kde H je hašovací funkce [8].

Odolností vůči modifikaci vzoru se požaduje, aby prakticky bylo téměř nemožné po vypočtení haše z prvního vzoru nalézt druhý vzor, který by měl stejný haš [8].

[2] Často označován anglicky hash

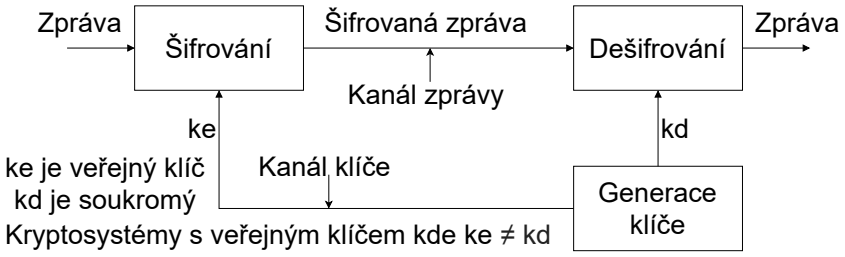
Při odolnosti vůči kolizím by mělo být téměř nemožné najít dva rozdílné vzory pro které by vycházel stejný haš [8].

Pro splnění efektivnosti by daná hašovací funkce měla být realizovatelná v polynomiálním čase, nejlépe však v lineárním [10].

Nejčastější využití hašovacích funkcí nalezneme v digitálních podpisech, kryptografických aplikacích generujících pseudonáhodné data a asymetrických kryptosystémech. V digitálních podpisech je hašovací funkce použita k vytvoření otisku zprávy, který slouží k ověření autentičnosti a integritě. Generace pseudonáhodných dat se využívá ve spoustě případů od autentifikačních protokolů po protokoly elektronické komerce. V asymetrických kryptosystémech se používá k ověření správnosti šifrovaného textu. Tento mechanizmus je nezbytný k zabezpečení ochrany proti útokům [10].

2.4 Asymetrická kryptografie

Asymetrická kryptografie někdy nazývaná kryptografie s veřejným klíčem používá dva rozdílné klíče. Tyto klíče se nazývají veřejný klíč, označovaný jako „public key“ a soukromý klíč, označovaný jako „private key“. Veřejný klíč se používá k šifrování zprávy a soukromý klíč k dešifrování zašifrované zprávy pomocí veřejného klíče. Hlavní výhodou této metody je, že příjemce vypočítá veřejný a soukromý klíč, ale sdílí pouze veřejný. Odesílatel pomocí asymetrické šifry a veřejného klíče šifruje zprávu, kterou následně pošle příjemci, který vydal veřejný klíč. Příjemce pomocí soukromého klíče dešifruje přijatou zprávu. Výhoda této metody spočívá v tom, že při odeslání špatnému příjemci nebo odcizení zprávy by neměl nesprávný příjemce dešifrovat zprávu bez znalosti soukromého klíče. Tyto dva klíče jsou propojené složitými matematickými výpočty, kdy veřejný klíč nenese informace o soukromém klíči. Z tohoto důvodu při požadavcích kladených na bezpečnost v dnešní době v reálném čase není možné vypočítat z veřejného klíče soukromý klíč. Nejznámější a nejpoužívanější šifrou je RSA (Rivest–Shamir–Adleman) [11]. Princip asymetrického kryptosystému je zobrazen na obrázku 2.2.



Obr. 2.2: Princip asymetrického kryptosystému [10].

2.5 Symetrická kryptografie

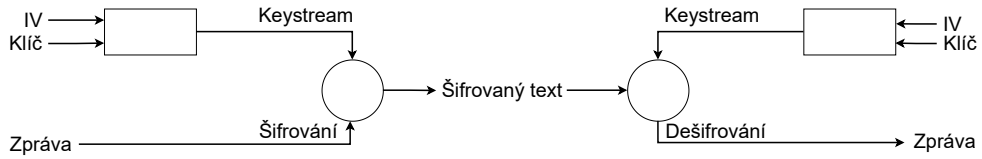
Symetrická kryptografie se odlišuje od asymetrické kryptografie počtem klíčů, kdy symetrická kryptografie využívá pouze jednoho soukromého klíče, který slouží pro šifrování a dešifrování. Tento klíč si obě dvě strany, které chtějí komunikovat, musejí vyměnit. Problém může nastat při ustanovení a výměně klíče, kde lze klíč jednoduše odchytit pokud spojení není zabezpečené. Bezpečná výměna a ustanovení klíče může proběhnout fyzickou výměnou, která v moderní době není možná v globálním měřítku. Při ustanovení a výměně klíče se přes nezabezpečenou síť využívá asymetrické kryptografie. V některých případech symetrické algoritmy využívají dva klíče – jeden k šifrování a druhý k dešifrování. Při odcizení šifrovacího klíče lze jednoduše vypočítat dešifrovací klíč a naopak. Jelikož se při vytváření šifry počítá, že algoritmus na ustanovení klíčů je veřejně známý, měl by být rozsah klíčů velmi velký. Pokud je rozsah klíčů malý, lze jej prolomit hrubou silou. Symetrické šifry dělíme na proudové a blokové [11].

2.5.1 Proudové šifry

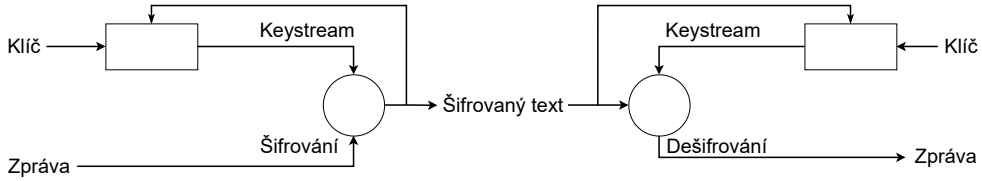
Mezi hlavní výhody proudových šifer patří bezchybovost nebo velice malá chybovost. Tyto šifry musí zpracovávat proud bitů a každý bit se šifruje samostatně. Oproti blokovým šifrám jsou rychlejší a méně hardwarově náročné. Lze je rozdělit na dva typy. Prvním typem jsou synchronní a druhým jsou asynchronní. Rozlišujeme je podle generace takzvaného keystreamu, což je generující klíčový proud znaků [12].

U synchronních proudových šifer nezáleží na šifrovaném a nešifrovaném textu při generaci keystreamu. Generace keystreamu je pouze závislá na klíči a šifrovacím algoritmu, ke kterému se u moderních šifer přidává ještě inicializační vektor (IV) [13]. Princip je zobrazen na obrázku 2.3. Jedním z požadavků na takovou šifru je synchronizace odesílatele a příjemce, to znamená, že každá strana musí mít stejný stav algoritmu a sdílený klíč. Výhodou je, že dojde-li při přenosu šifrované zprávy k záměně znaku, zbytek zprávy nebude ovlivněn. Z čehož plyne, že synchronní proudové šifry jsou vhodné pro použití tehdy, když na přenosové lince dochází k velké chybovosti. Při ztrátě synchronizace odesílatele a příjemce nelze zprávu dešifrovat [12].

U asynchronních proudových šifer oproti synchronním šifrám je keystream generován šifrovacím algoritmem za pomoci klíče a předchozích šifrovaných znaků zprávy. Princip je zobrazen na obrázku 2.4. Stejně jako u synchronních šifer by měl být příjemce a odesílatel synchronizován. Dojde-li však k desynchronizaci, měla by se šifra sama po několika šifrovaných znacích synchronizovat. Pokud při přenosu nastane chyba, při dešifrování několika následujících znaků bude dešifrování provedeno chybně, ale zbytek zprávy bude dešifrován v pořádku, jestliže nenastanou další chyby [12].



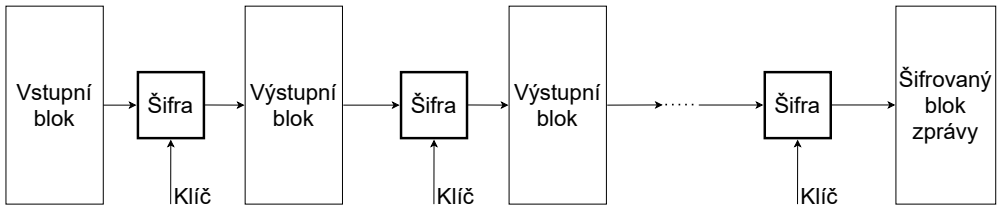
Obr. 2.3: Princip šifrování a dešifrování synchronních proudových šifer [13].



Obr. 2.4: Princip šifrování a dešifrování asynchronních proudových šifer [14].

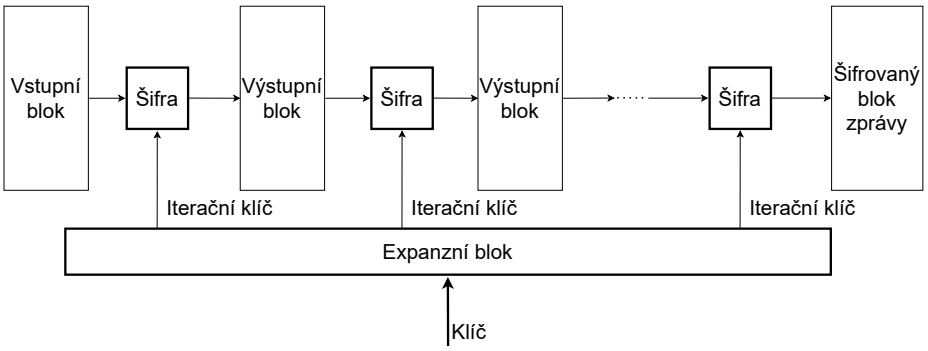
2.5.2 Blokové šifry

Blokové šifry na rozdíl od proudových provádí šifrování po blocích, kde každý blok je stejně dlouhá n-tice bitů. Moderní blokové šifry jsou založeny na principu kaskádových šifer. Kaskádové šifry fungují na základě řetězení několika šifer. Do následující šifry je vstupním blokem výstupní blok z předcházející šifry. Poslední výstupní blok je výsledný šifrovaný blok zprávy. Princip je zobrazen na obrázku 2.5. V blokových šifrách se používá princip iterované šifry, která se dělí na Feistelovu síť a substitučně-permutační síť [8].



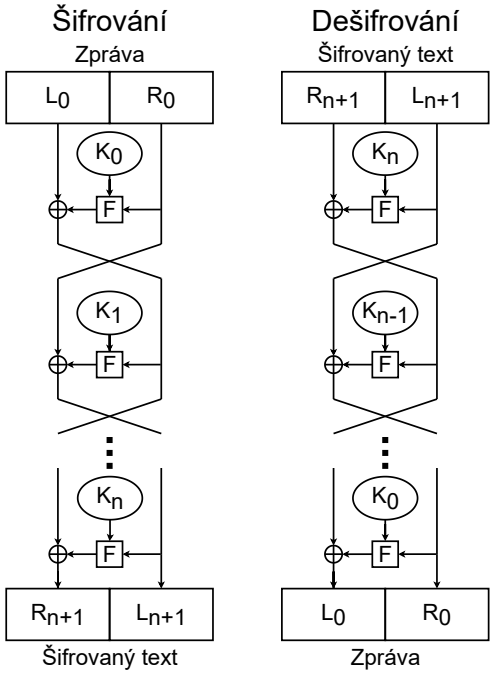
Obr. 2.5: Princip kaskádové šifry [8].

Iterované šifry využívají princip kaskádových šifer, kdy se používá zřetězení stejné šifry po několik iterací. Princip je zobrazen na obrázku 2.6. Jako šifra pro iteraci je nejčastěji využívána některá z jednoduchých šifer. Z klíče se odvodí několik iteračních klíčů v expanzním bloku. Tyto klíče jsou následně použity postupně pro každou iteraci. Jednoduchá šifra společně s klíčem pomocí vhodných blokových operací jako jsou permutace, substituce a aritmetické operace, šifrují data do výstupního bloku, který je následně použit jako vstup do další iterace [8].



Obr. 2.6: Princip šifrování iteračních kaskádových šifer [8].

Prvním typem iterované šifry je Feistelova síť zobrazena na obrázku 2.7. Tento typ šifry funguje na principu rozdělení vstupního bloku na levou a pravou polovinu. Pravá polovina a iterační klíč vstupují do konverzní funkce, která má výstup o délce poloviny vstupního bloku. Výsledkem iterace je blok o velikosti vstupního bloku, kde novou levou stranou je předcházející pravá strana a pravou stranou je předcházející levá strana. Jak je patrné, tak šifrována byla pouze levá strana, z čehož vyplývá, že k zašifrování vstupního bloku jsou potřeba nejméně dvě iterace. Z tohoto důvodu bývá počet iterací sudý a dostatečně velký do té míry, aby se zabezpečila difuze šifry [8].



Obr. 2.7: Princip šifrování a dešifrování Feistelovy sítě [15].

Druhým typem jsou substitučně-permutační sítě zkráceně SPN. Tento typ šifer je založen na substituci a následné permutaci vstupních bitů. Při substituci se využívá takzvaných S-boxů, které pracují s částí zprávy, nejčastěji o velikosti 4 a 8 bitů, kdy se posloupnost těchto bitů nahradí za jinou posloupnost. Permutace se realizuje pomocí takzvaných P-boxů. Permutace se snaží dosáhnout co největší difuze, aby bylo ovlivněno maximální množství S-boxů [22].

Nejčastějšími operacemi používané v substitučně-permutačních sítích jsou:

- SubByte – nelineární substituce bitů, které operují nezávisle na sobě a jsou zaměňovány dle předem dané tabulky substituce, která je zároveň klíčem.
- ShiftRow – v této operaci se bity posouvají v řádku matice do stran a každý řádek se posouvá o jiný počet míst.
- MixColumns – spočívá v záměně sloupců, následně je každý sloupec vynásoben polynomem o stejné hodnotě.
- AddRoundKey – každý bit je zkombinován s iteračním klíčem, kdy po kombinaci bitů zprávy a iteračního klíče dostaneme výsledný šifrovaný text [12].

2.6 Provozní režim blokových šifer

Provozní režim pro blokové šifry je nazýván také jako operační mód a určuje jakým způsobem jsou šifrována a dešifrována data o větší délce bloku než podporuje šifrovací algoritmus. Existuje 5 základních provozních režimů. Těmito režimy jsou režim kódové knihy (ECB), řetězení šifrových bloků (CBC), šifrová zpětná vazba (CFB), výstupní zpětná vazba (OFB) a čítačový režim (CTR). Dále existuje Galois/counter mód (GCM), který je složením CTR a Galoisova autentifikačního módu a nebude zde popsán dopodrobna. Jelikož vstupní data nebývají často dělena vstupní n-ticí blokové šifry beze zbytku, musí se doplnit o zbývající počet bitů. K tomuto účelu bylo vytvořeno několik standardů, sloužících pro vytvoření tohoto paddingu. Nejznámějšími standardy jsou PKCS7, ISO/IEC 7816-4 a ANSI X9.23. Nevýhodou při aplikaci paddingu je nutnost ho použít vždy tehdy, i když se v dané zprávě nepotřebuje využít. V tomto případě se musí doplnit celý n-bitový blok, aby se zajistila jednotnost pro dešifraci [16][17][18].

Padding pomocí standardu PKCS7 funguje na principu doplnění zbývajících bitů pomocí n-tice bytů, kdy jejich hodnota odpovídá počtu přidaných bytů. Padding pomocí standardu ISO/IEC 7816-4 doplňuje zbývající byty doplněním jednoho bytu s hodnotou 0x80 a zbytek bytů nulovou hodnotou. Padding pomocí standardu ANSI X9.23 je definován na doplnění nulových bytů, kdy poslední má hodnotu odpovídající počtu přidaných bytů[17]. Příklad pro tyto standardy je uveden v tabulce 2.1, u kterého se provádí doplnění na 8 bytů.

Tab. 2.1: Doplnění paddingu na 8 bytů.

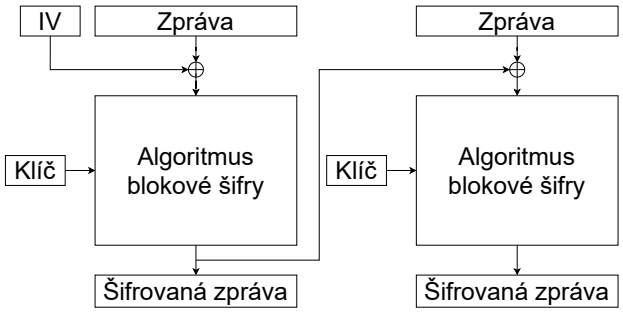
Název standardu	Vstupní počet bytů	Doplnění paddingu
PKCS7	ABCDEF	ABCDEF0505050505
ISO/IEC 7816-4	ABCDEF	ABCDEF8000000000
ANSI X9.23	ABCDEF	ABCDEF0000000005

2.6.1 Kódová kniha

Režim kódové knihy označován jako ECB z anglického electronic codebook je nejjednoduší operační mód. Prvním krokem je doplnění paddingu zprávy. Ve druhém kroku je zpráva z rozdělena na n-bitové bloky z_i. Třetím krokem je individuální šifrování těchto bloků z_i vybranou blokovou šifrou. Dešifrování probíhá stejně jako šifrování. Nevýhoda tohoto operačního módu je nemožnost skrytí vzorů, které se objevují ve zprávě. Z tohoto důvodu není doporučeno uvedený operační mód používat. ECB umožňuje paralelizovat, a to jak šifrování tak i dešifrování [16].

2.6.2 Řetězení šifrových bloků

Řetězení šifrových bloků označováno jako CBC z anglického cipher block chaining je jedním z nejpoužívanějších operačních módů. V prvním kroku se aplikuje padding stejně jako v případě ECB. V druhém kroku je zpráva z rozdělena na n-bitové bloky z_i. Ve třetím kroku se využije operace XOR mezi šifrovaným blokem z_{i-1} a vstupním blokem z_i. Na výsledek po operaci XOR je následně aplikována bloková šifra. Pro první blok z_1 se zaměňuje z_{i-1} za IV (inicializační vektor). Tento princip je zobrazen na obrázku 2.8. Inicializační vektor je náhodná vstupní hodnota k zajištění rozdílnosti šifrované zprávy při stejné vstupní zprávě. Dále se inicializační vektor využívá také v operačních módech CFB, OFB a CTR. Dešifrování probíhá opačným způsobem jak šifrování a pro dešifrování prvního bloku musí být znám IV. CBC umožňuje pouze paralelizovat dešifrování [16].

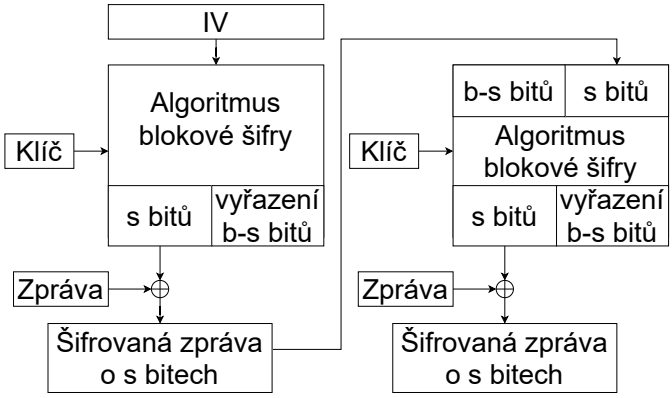


Obr. 2.8: Princip šifrování pomocí řetězení šifrových bloků [19].

2.6.3 Šifrová zpětná vazba

Šifrová zpětná vazba označována jako CFB z anglického cipher feedback umožňuje změnit blokovou šifru do podoby asynchronní proudové šifry. Vstupem pro šifrování je výsledek operace XOR šifrovaného vstupního bloku a zprávy z_{-1}. Pro první vstupní blok je využit IV, který je samostatně šifrován. Operační mód dále často používá k operaci XOR menší část zprávy než je velikost bloku. Tento přídavný mechanizmus zvýší počet potřebných provázání a je k němu potřeba s, kde $s \in \mathbb{N}^+, 1 \leq s \leq b$ a b je velikost bloku blokové šifry. Pro CFB módy využívající tento mechanizmus je šifrování popsáno následovně [20].

Vstupem prvního bloku stále zůstává IV. Ze zašifrovaného vstupního bloku se použije s nejvíce signifikantních bitů, na které se využije operace XOR se stejným počtem bitů zprávy. Tento výsledek je první částí šifrované zprávy. Vstupem dalšího bloku je spojení s bitů šifrované zprávy a $b-s$ nejméně signifikantních bitů IV. Toto spojení je dalším vstupním blokem. Tento postup je zobrazen na obrázku 2.9. CFB umožňuje pouze paralelizovat dešifrování a nepotřebuje využít paddingu [20].

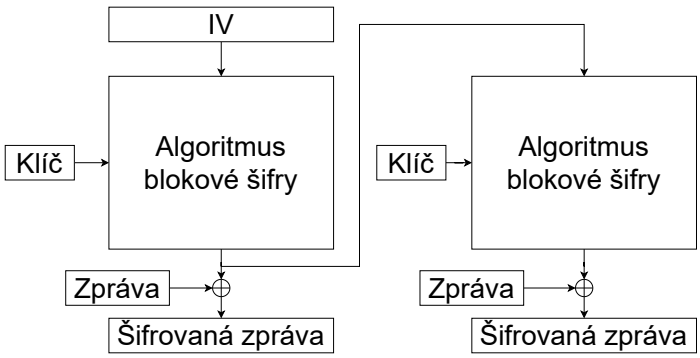


Obr. 2.9: Princip šifrování pomocí šifrové zpětné vazby [20].

2.6.4 Výstupní zpětná vazba

Výstupní zpětná vazba označována jako OFB z anglického output feedback mění blokovou šifru do podoby synchronní proudové šifry. Funguje na stejném principu jako operační mód CFB v jeho základní variantě bez dělení na s-bitové části. Hlavním rozdílem je výpočet vstupního bloku, kdy jako vstup je použit výstupní blok bez použití operace XOR se zprávou. Vstup pro první vstupní blok je znovu použit IV a výsledná šifrovaná zpráva se získá provedením operace XOR mezi výstupním blokem a zprávou. Tento postup je zobrazen na obrázku 2.10. Jestliže poslední blok

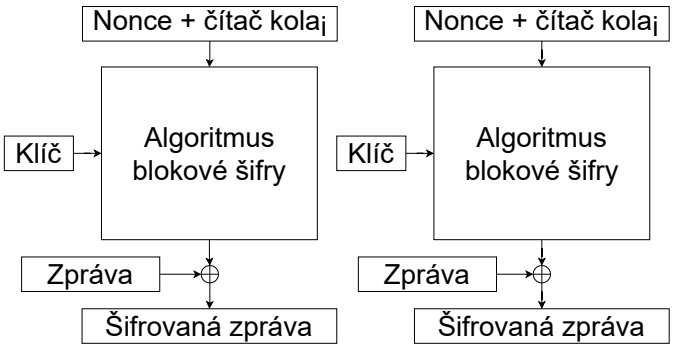
má menší velikost u, kde $u \in \mathbb{N}^+, 1 \leq u \leq b$ a b je velikost bloku blokové šifry, odstraní se $u - b$ nejméně signifikantních bitů z výstupní zprávy. Pro výpočet poslední části šifrované zprávy je využita operace XOR mezi zbylými bity výstupní zprávy a poslední částí zprávy [20].



Obr. 2.10: Princip šifrování pomocí výstupní zpětné vazby [20].

2.6.5 Čítačový režim

Čítačový režim označován jako CTR z anglického counter mění stejně jako operační mód OFB blokovou šifru na synchronní proudovou šifru. Vstupním blokem je takzvaná hodnota „nonce“, kterou nejčastěji bývá IV. K této hodnotě je přičteno i, které je hodnotou pořadí bloku. Výsledek je následně modulován 2^n, kde n je bitová velikost vstupního bloku a tímto se zajistí vždy validní vstup o konstantní délce. Vstupní blok je poté šifrován blokovou šifrou. Na šifrovaný výstupní blok společně s blokem zprávy je použita operace XOR. Výsledek této operace je šifrovaný text. Při kratším posledním bloku zprávy se postupuje stejně jako v případě OFB. CTR umožňuje paralelizovat jak šifrování tak i dešifrování [16].



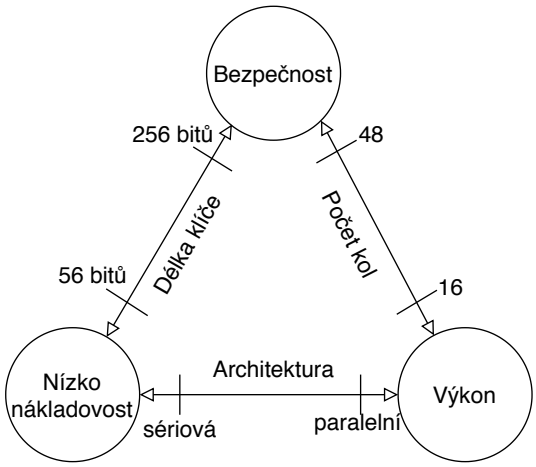
Obr. 2.11: Princip šifrování pomocí čítačového režimu [21].

3 Lehká kryptografie

Kvůli potřebám šifrování na výkonově omezených zařízeních jako jsou mikrokontroléry, čipy využívající identifikace na rádiové frekvenci (RFID) a jim podobná zařízení, se vytvořila nová kategorie šifer nazývaná lehká kryptografie. Lehká kryptografie je relativně mladé odvětví, ve kterém se prolíná kryptografie, informatika a elektroinženýrství. Z důvodu omezeného výkonu nelze na tyto zařízení implementovat většinu standardních kryptografických algoritmů [22][23].

V posledních letech se zvýšila implementace kryptografie na programovatelných hradlových polích (FPGA) kvůli jejich čím dál většímu komerčnímu rozšíření. Hlavním důvodem, který k tomu vedl, je jejich výkon oproti procesorům a možnost přeprogramování na rozdíl od integrovaných obvodů určených pro aplikaci (ASIC) čipů. Lehká kryptografie se implementovala převážně na ASIC čipy, které byly výhodné, pokud se vytvářely ve velkém množství. Hlavní nevýhodou ASIC čipů je nemožnost přeprogramování po jejich výrobě. Z tohoto důvodu se čím dál více snaží prosadit implementace šifer lehké kryptografie na FPGA čipy, které jsou nízkoenergetické a v dnešní době cenově dostupné. Přeprogramování FPGA umožňuje opravit případné nově nalezené chyby v šifrách po implementaci. Poslední výhodou je možnost přidat kryptografický IP (Intelectual Property) blok do návrhu nebo upravit aktuální návrh ke zmenšení celkové spotřeby plochy a tím i nákladů s tím spojených [24][25].

Při návrhu algoritmů lehké kryptografie je nutno udělat kompromis mezi bezpečností, náklady a výkonem [22]. Tento kompromis lze vidět na obrázku 3.1.



Obr. 3.1: Kompromis při návrhu algoritmu [23].

Nejčastěji bývají splněny pouze dva ze tří požadavků na návrh algoritmu, a to buď bezpečnost a nízkonákladovost, bezpečnost a výkon nebo výkon a nízkoná-

kladovost. Docílení všech tří požadavků tak, aby byly správně optimalizované, bývá velmi náročné. Například vysokého výkonu na hardwaru a bezpečnosti lze dosáhnout zřetězením architektur, které zakomponovávají několik protiopatření proti útokům postranními kanály. Výsledný návrh bude mít vysoké plošné požadavky, což koreluje s vysokými náklady. Naopak lze navrhnout algoritmus, který je bezpečný a má malé náklady. Tento algoritmus bude omezen výkonem [22][23].

Pro zařízení, které využívají lehké kryptografie, byl stanoven požadavek na minimální bezpečnost o velikosti 80 bitů. Tato délka se zdá být nedostatečná, ale vzhledem k informaci, která je chráněna, je dostatečná. Při pokusu o prolomení takového zabezpečení by útočník musel vynaložit velké úsilí nebo finanční obnos. Zisk z tohoto pokusu by byl ve většině případů zcela neadekvátní [26].

Lehká kryptografie využívá hašovací funkce, proudové a blokové šifry, které byly speciálně pro tuto potřebu vyvinuty [26].

3.1 Hašovací funkce lehké kryptografie

V lehké kryptografii hašovací funkce fungují na stejném principu jak je uvedeno v kapitole 2.3 Hašovací funkce. Jediný rozdíl je v bitové délce výstupu. V případě hašovacích funkcí, které se používají na zařízeních u kterých není omezený výkon, se využívá výstup hašovací funkce o velikosti 128 bitů a více. Ve většině případů se začaly používat hašovací funkce typu SHA-256 a SHA-512, které mají délku 256 bitů, respektive 512 bitů. Hašovací funkce z rodiny SHA jsou nejpoužívanější hašovací funkce v kryptografii [27].

Lehká kryptografie používá hašovací funkce o velikosti výstupu 80 bitů a více. Nejznámější hašovací funkcí je PHOTON, která má více variant délky výstupu. Tyto délky výstupu jsou v rozsahu od 80 bitů do 256 bitů. Další známou hašovací funkcí je SPONGENT, která nabízí varianty výstupu od 88 bitů do 256 bitů [27].

3.2 Proudové šifry lehké kryptografie

Proudové šifry pro lehkou kryptografii fungují na stejném principu, který je popsán v kapitole 2.5.1 Proudové šifry. Proudových šifer pro lehkou kryptografii není navrženo mnoho. Nejpoužívanější šifrou je šifra A5/1, jejíž využití se vyčísluje v miliardách. Jedním z důvodů proč jich není vytvořeno mnoho, může být, že stávající proudové šifry jsou několikanásobně méně náročné na implementaci. Z tohoto důvodu lze stávající šifry uzpůsobit jejich implementaci nebo šifry samotné tak, aby je bylo možné použít na výpočetně omezených zařízeních [22].

Mezi zástupce proudových šifer patří E_0. Tato šifra se často vyskytuje ve sféře mobilních telefonů. Využívá se pro přenos paketů, které udělují důvěrnost při přenosu protokolem Bluetooth. Algoritmus používá klíč, který má délku 128 bitů. Dalším zástupcem je šifra A5/1. Tato šifra se využívá v sítích GSM[1] a je i součástí jejich standardu. Využívá 64bitový klíč a 22bitový inicializační vektor. Dále byla v roce 2007 představena šifra Grain, která se aplikuje hlavně u zařízení, které mají omezenou paměť a spotřebu energie. Funguje na principu dvou posuvných registrů a nelineární funkci, kdy její klíč má délku 80 bitů. Existuje i verze, která využívá klíč o délce 128 bitů [22].

3.3 Blokové šifry lehké kryptografie

Princip blokových šifer byl detailně probrán v kapitole 2.5.2 Blokové šifry. Nejznámější blokovou šifrou, která je založena na principu Feistelovy sítě je šifra DES. Z šifry DES vychází čínská šifra LBlock, která je jednou z hlavních šifer používaných v lehké kryptografii a byla navržena pro zařízení s nedostatkem výpočetního výkonu. Nejznámější blokovou šifrou, která je založena na principu subtitučně-permutační sítě, je šifra AES. Z této šifry vznikla jedna z nejznámějších lehkých šifer – evropská šifra PRESENT. Šifra PRESENT je podobně jako LBlock navržena z ohledem na zařízení, která mají omezené prostředky výpočetního výkonu [28].

3.3.1 PRESENT

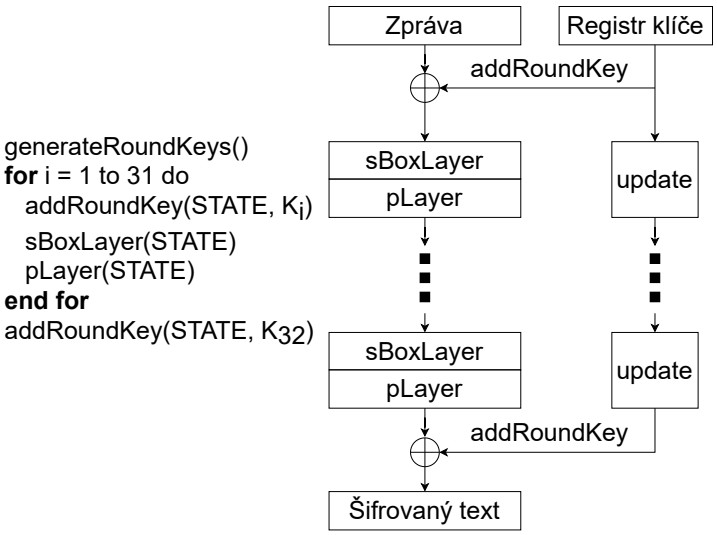
Šifra PRESENT využívá bloky o délce 64 bitů a podporuje velikost klíče 80 a 128 bitů. Ve standardu tvůrci navrhují využití 80bitové varianty. Počet kol opakování šifry je 31, v každém kole je použit klíč určený pro dané kolo, který provádí exkluzivní disjunkci (XOR) se vstupním 64bitovým blokem. Následně po použití operace XOR se provádí lineární bitová permutace a nelineární substituce. Nelineární substituce využije jednoho 4bitového S-boxu, který je použit paralelně 16 krát v každém kole. Výsledkem bude šifrovaný blok textu [29].

Substituční vrstva – využívá 4bitové S-boxy, které paralelně nahrazuje. V tabulce 3.1 je uveden postup nahrazovaní bitů. Na obrázku 3.2 je tato vrstva označena pod názvem sBoxLayer [29].

Tab. 3.1: Tabulka pro nahrazení S-boxů [29].

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

[1] Globální systém pro mobilní komunikaci



Obr. 3.2: Algoritmický princip šifry PRESENT [29].

Permutační vrstva – v následují tabulce 3.2 je uveden postup permutace, kdy bit i je předchozí pozice bitu a *P(i)* uvádí novou pozici bitu. Tato vrstva je označena na obrázku 3.2 jako pLayer [29].

Tab. 3.2: Permutační tabulka [29].

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(i)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(i)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(i)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(i)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Key schedule – podporuje 80bitový nebo 128bitový klíč, v následujícím postupu se zaměříme na 80bitový klíč. Klíč dodaný uživatelem je uložen v registru klíče K, je reprezentován jako posloupnost bitů $k_{79}\, k_{78} \ldots k_0$. V kole i se používá klíč, ve kterém je použito prvních 64 bitů z levé strany uložených v registru K. V kole i se tedy použije klíč:

$$K_i = k_{63}\, k_{62} \ldots k_0 = k_{79}\, k_{78} \ldots k_{16}$$

Po extrakci klíče se registr K upraví:

$$k_{79}\,k_{78}\ldots k_1\,k_0 = k_{18}\,k_{17}\ldots k_{20}\,k_{19}$$
$$k_{79}\,k_{78}\,k_{77}\,k_{76} = S(k_{79}\,k_{78}\,k_{77}\,k_{76})$$
$$k_{19}\,k_{18}\,k_{17}\,k_{16}\,k_{15} = (k_{19}\,k_{18}\,k_{17}\,k_{16}\,k_{15}) \oplus \mathsf{round\_counter}$$

Registr klíče K je posunut o 61 bitů doleva, první čtyři bity z levé strany se použijí k substituci s S-boxem. V poslední části se využije operace XOR mezi bity $k_{19}\,k_{18}\,k_{17}\,k_{16}\,k_{15}$ z registru klíče K a hodnotou round_counter, která udává počet kol i které proběhly [29].

3.3.2 LBlock

Šifra LBlock využívá bloky o velikosti 64 bitů s klíčem o délce 80 bitů. Počet kol opakování je 32. V další části textu budou použity tyto notace [30]:

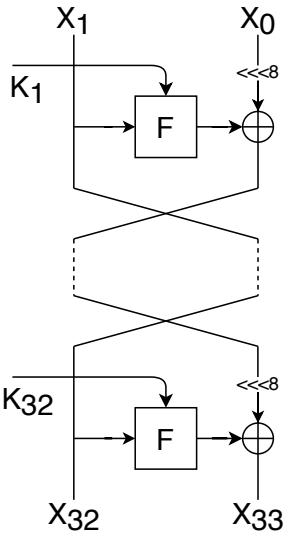
M	64 bitová zpráva
C	64 bitový šifrovaný text
K	80 bitový klíč
K_i	32 bitový klíč pro kolo i
F	funkce pro kolo
S	S-boxová vrstva
P, P_1	permutační operace pro 32 bitů
$<<< 8$	bitový posun o 8 bitů vlevo

Algoritmus šifrování je znázorněn na obrázku 3.3. Na začátku šifrování se rozdělí zpráva M na 2 části X_1 a X_0, každá z částí má velikost 32 bitů. Po posledním kole se výsledné dva bloky X_{32} a X_{33} spojí do jednoho 64bitového bloku, který obsahuje šifrovaný text C [30].
Průběh jednoho kola i lze vyjádřit jako:

$$X_i = F(X_{i-1}, K_{i-1}) \oplus (X_{i-2} <<< 8)$$

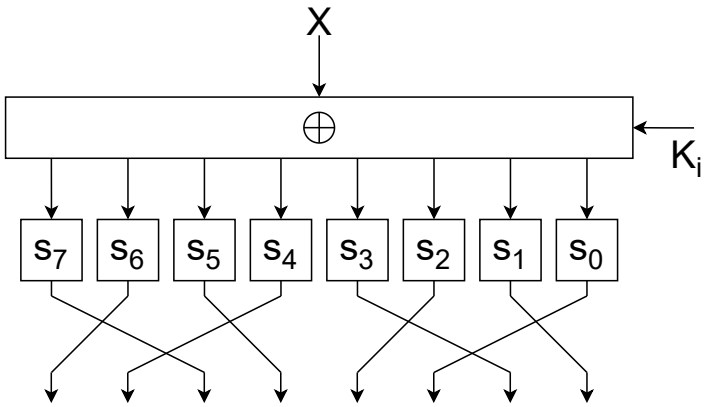
Funkci kola F lze vyjádřit jako:

$$\{0,1\}^{32} \times \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$$
$$(X, K_i) \longrightarrow U = P(S(X \oplus K_i))$$



Obr. 3.3: Algoritmický princip šifry LBlock [30].

Funkce konfuze S – obsahuje 8 paralelně poskládaných 4×4 S-boxů, jejichž obsah je zobrazen v tabulce 3.3, kde jsou označeny $s_0 \dots s_7$. Každý blok se rozdělí na části o velikosti 4 bitů. Na každou část je použit S-box. Tímto lze zaručit, že každý bit z 32bitového bloku bude ovlivněn [30].



Obr. 3.4: Funkce pro kolo šifry LBlock [30].

Funkce difuze P – je permutace osmi 4 bitových slov, které jsou výstupem funkce S.
Key schedule – 80bitový klíč K je uložen v registru klíče K_r jako posloupnost bitů $K = k_{79}\, k_{78} \dots k_0$. Klíč pro kolo K_i je vytvořen z prvních 32 bitů z levé strany, kde se následovně K_r posune o 29 bitů vlevo. V další části na prvních 8 bitů z levé strany jsou použity dva S-boxy s_8 a s_9, které lze dohledat v tabulce 3.3. V poslední částí se využije operace XOR mezi bity $k_{50}\, k_{49}\, k_{48}\, k_{47}\, k_{46}$ a hodnotou i, která udává hodnotu

aktuálního kola. Výstupem je prvních 32 bitů z levé strany registru K jako klíč K_{i+1}. Postup transformace klíče pro kolo i je zobrazen níže, kde i je pro kola 1 až 31 [30].

(A) $K = k_{79}\, k_{78} \ldots k_{49}\, k_{48}$

(B) $K <<< 29$

(C) $k_{79}\, k_{78}\, k_{77}\, k_{76} = s_9[k_{79}\, k_{78}\, k_{77}\, k_{76}]$

(D) $k_{79}\, k_{78}\, k_{77}\, k_{76} = s_8[k_{79}\, k_{78}\, k_{77}\, k_{76}]$

(E) $k_{50}\, k_{49}\, k_{48}\, k_{47}\, k_{46} \oplus i$

(F) $K_{i+1} = k_{79}\, k_{78} \ldots k_{49}\, k_{48}$

Tab. 3.3: S-box tabulka pro šifru LBlock [30].

s_0	14	9	15	0	13	4	10	11	1	2	8	3	7	6	12	5
s_1	4	11	14	9	15	13	0	10	7	12	5	6	2	8	1	3
s_2	1	14	7	12	15	13	0	6	11	5	9	3	2	4	8	10
s_3	7	6	8	11	0	15	3	14	9	10	12	13	5	2	4	1
s_4	14	5	15	0	7	2	12	13	1	8	4	9	11	10	6	3
s_5	2	13	11	12	15	14	0	9	7	10	6	3	1	8	4	5
s_6	11	9	4	14	0	15	10	13	6	12	5	7	3	8	1	2
s_7	13	10	15	0	14	4	9	11	2	1	8	3	7	5	12	6
s_8	14	9	15	0	13	4	10	11	1	2	8	3	7	6	12	5
s_9	4	11	14	9	15	13	0	10	7	12	5	6	2	8	1	3

4 Výběr algoritmu

V této kapitole je popsán výběr vhodného algoritmu k implementaci na FPGA. První část se zaměří na výběr podle typu šifer tak, aby byla vybrána šifra, která je vhodná pro šifrování souborů a dat uložených na médiích. Druhá část popisuje výběr konkrétní šifry z vybraného typu šifer. Výběr se zabývá jak jejich hardwarovými požadavky, tak i základními požadavky na bezpečnost.

4.1 Výběr typu šifry

Výběr probíhal ze tří typů šifer – hašovací funkce, proudové a blokové šifry. Postupně budou popsány klady a zápory každého typu šifer a nakonec bude uvedena vybraná šifra.

Hašovací funkce mají výhodu v jejich rychlosti a jednoduchosti. Další předností je, že jestliže je vstupní blok dat vždy různě dlouhý, výsledek hašovací funkce bude mít vždy stejnou výstupní délku. Hlavní nevýhoda spočívá v jednosměrnosti hašovacích funkcí. Jelikož je nemožné z hašovací funkce získat původní data, používají se nejčastěji pro ověření elektronických podpisů. Z důvodu jednosměrnosti jsou hašovací funkce pro potřeby této práce nevhodné, protože budou šifrována data, ze kterých by mělo být možné získat původní hodnotu, která byla zašifrována.

Výhodou proudových šifer je jejich malá hardwarová náročnost, protože šifrují data bit po bitu. Hlavní nevýhodou je možnost změny dat při narušení přenosu. Šifry tohoto typu se nejčastěji využívají při přenosu dat přes síť, kde se přenáší data mezi jednotlivými body. Jelikož při implementaci šifry na FPGA se nepočítá se šifrováním dat síťového provozu, označil bych tento typ šifer také za nevhodný.

U blokových šifer je hlavní výhodou, že se šifrují po určitém stejném bloku dat, kde každý bit v těchto datech je šifrován mezi několik bitů ve výsledném šifrovaném textu. Další nesporný klad spočívá v problematické záměně dat v šifrované zprávě. Při záměně části bitů v šifrovaném textu lze po dešifrování rozpoznat, jestli s daty bylo manipulováno. Jedinou možností útočníka by pak bylo upravená data šifrovat pomocí stejného algoritmu a klíče. Nevýhodou je pomalejší šifrování na rozdíl od výše zmíněných typů. Chyba při šifrování neovlivní pouze daný bit nebo bity, ale celý šifrovaný blok dat. Tyto šifry jsou vhodné pro šifrování dat, u kterých je známa délka.

Pro implementaci šifrování souborů uložených na médiích lze z těchto tří popsaných typů a z výše uvedených kladů a záporů říci, že pro šifrování souborů, u kterých známe předem velikost, vychází blokové šifry jako nejvhodnější. Hašovací funkce nemohou být použity kvůli své jednosměrnosti, proudové šifry nejsou často užívány k šifrování souborů, které se nepřenáší přes síť.

4.2 Výběr šifry

Blokových šifer pro lehkou kryptografii je velké množství, výběr byl proveden z následujících šifer – LBlock, LED, mCRYPTON, PRESENT, SIMON. Tyto šifry jsou nejznámějšími zástupci blokových šifer pro lehkou kryptografii.

Tab. 4.1: Porovnání blokových šifer z hlediska výkonu [22].

Název	Oblast [GE]	Propustnost při 100 kHz [kb/s]	Efektivnost [bps/GE]
LBlock	1320	200,00	48,69
LED-64	966	5,10	5,28
LED-128	1265	3,40	2,69
mCRYPTON-64	2420	492,31	203,43
mCRYPTON-96	2681	492,31	183,63
mCRYPTON-128	2949	492,31	119,84
PRESENT-80	1570	200,00	127,39
PRESENT-128	1884	200,00	106,16
SIMON 32/64	566	22,20	39,22
SIMON 48/96	763	15,00	19,66
SIMON 64/96	838	17,80	21,24
SIMON 64/128	1000	16,70	16,70
SIMON 96/96	984	14,80	15,04
SIMON 128/128	1317	22,90	17,31
SIMON 128/256	1883	21,10	11,21

V tabulce 4.1 jsou vypsány šifry a jejich varianty dle alfabetické posloupnosti. Gate equivalent (GE) je jednotka, která udává počet požadovaných integrovaných obvodů (IC). Je odvozena dělením plochy IC s logickou oblastí negovaného logického součinu (NAND) s nejnižší řídící silou.

Při srovnání hodnot GE jsou na implementaci nejvhodnější šifry SIMON, kdy čísla uvedená v názvu udávají velikost bloku a velikost klíče. Pro upřesnění šifry SIMON 32/64 lze uvést, že zabírá na obvodech nejméně plochy (566 GE), má velikost bloku 32 bitů a velikost klíče 64 bitů. Nejvíce náročnou šifrou je mCRYPTON ve všech jejich variantách. Navzdory hardwarové náročnosti je její nespornou výhodou propustnost, která má hodnotu 492,31 kb/s pro všechny její varianty. Nejmenší propustnost má šifra LED, kterou následuje šifra SIMON. Obě tyto šifry mají podobné požadavky na využitou logickou oblast, ale šifra SIMON je všech kategoriích o poznání lepší. K propustnosti se váže i efektivnost, kdy pozice nejlepších a nejhorších šifer zůstávají nezměněné. Nejvyváženějšími šiframi jsou LBlock a PRESENT.

Tyto šifry nemají vysoké nároky na použitou logickou oblast, ale zároveň dosahují několikanásobné propustnosti oproti šifrám s podobnou hodnotou GE. Při porovnání efektivnosti šifer LBlock a PRESENT je šifra PRESENT značně efektivnější než LBlock, i když obě dvě mají stejnou propustnost. Tento rozdíl je tvořen v odlišnosti návrhu šifer, jelikož PRESENT využívá substitučně-permutační síť a LBlock Feistelovu síť.

Tab. 4.2: Porovnání blokových šifer z hlediska bezpečnosti [22].

Název šifry	Délka bloku [bit]	Délka klíče [bit]	Počet kol	Počet překonaných kol	
LBlock	64	80	21	23	72 %
LED-64	64	64	32	14	54 %
LED-128	64	64	48	23	67 %
mCRYPTON-64	64	64	13	7	54 %
mCRYPTON-96	64	96	13	7	54 %
mCRYPTON-128	64	128	13	7	54 %
PRESENT-80	64	80	31	26	84 %
PRESENT-128	64	128	31	26	84 %
SIMON 32/64	32	64	32	23	72 %
SIMON 48/96	48	96	36	25	69 %
SIMON 64/96	64	96	42	30	71 %
SIMON 64/128	64	128	44	31	70 %
SIMON 96/96	96	96	52	37	71 %
SIMON 128/128	128	128	68	49	72 %
SIMON 128/256	128	256	72	53	74 %

V tabulce 4.2 jsou porovnány šifry z pohledu bezpečnosti. Nejbezpečnější šifrou lze označit LED, u níž se podařilo překonat pouze polovinu z 32 kol pro její 64bitovou verzi klíče. Za druhou nejbezpečnější lze považovat šifru mCRYPTON, kde bylo překonáno 7 kol ze 13. Nejméně bezpečnou šifrou z vybraných je PRESENT, kdy zbývalo překonat 5 kol z 31, aby byla šifra prolomena. Šifry LBlock a SIMON se pohybují kolem hranice 70% překonaných kol. Vezmeme-li v úvahu, že z pohledu bezpečnosti je výhodnější delší klíč, jeví se šifra SIMON 128/256 jako nejvhodnější v kategorii podle délky klíče. Následují SIMON, PRESENT, mCRYPTON a LED v jejich variantách využívající 128bitového klíče. Jestliže spojíme tyto dvě informace dohromady, lze šifru mCRYPTON označit jako nejbezpečnější, po ní následuje šifra LED ve variantě 128 bitů. Zbytek šifer bych označil jako velmi podobných, kdy šifry SIMON s klíči o velikostech 256 a 128 jsou ze zbylých šifer nejbezpečnější. Za nejméně bezpečnou šifru lze považovat SIMON v 64bitové variantě.

Dále budeme z obou dvou srovnání vybírat nejvhodnější šifru, která je jak bezpečná tak dostatečně efektivní a zároveň nezabírá velkou logickou oblast. Jak bylo uvedeno v předchozí části, šifra SIMON zabírá nejmenší plochu, dá se považovat za bezpečnou, ale její efektivnost a propustnost zůstává velmi malá. Šifru mCRYPTON lze považovat za jednu z nejbezpečnějších, její propustnost a efektivnost je nejlepší z porovnávaných šifer, ale její logická oblast, kterou potřebuje k implementaci je nejméně dvojnásobná oproti ostatním implementacím. Šifra LED je podobná šifře SIMON v porovnávaných hodnotách s rozdílem bezpečnosti, kdy její 128bitovou variantu lze považovat za druhou nejbezpečnější. Šifra PRESENT společně s LBlock byly vybrány jako nejoptimálnější šifry z pohledu výkonu. Z hlediska bezpečnosti lze obě dvě šifry považovat za dostatečně bezpečné. Pokud porovnáme tyto dvě šifry vzájemně z pohledu výkonu, je jejich propustnost zcela stejná. Šifra PRESENT je lepší v kategorii efektivnosti, ale obě její varianty mají větší nároky na použitou logickou oblast. Z pohledu bezpečnosti mají obě dvě šifry délku bloku 64 bitů a jejich nejmenší varianty mají 80 bitů. Jedinou výhodou šifry LBlock je, že má méně překonaných kol než šifra PRESENT.

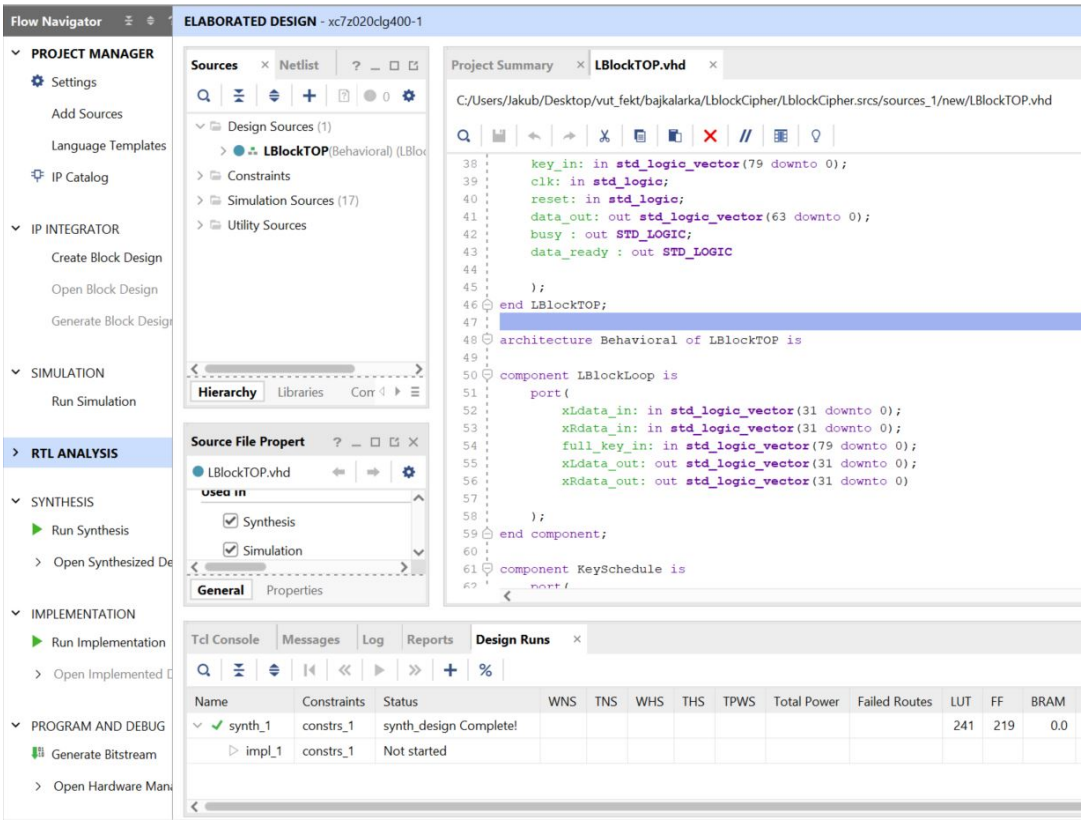
Porovnáním kladů a záporů všech šifer lze dojít k závěru, že konečné rozhodnutí bude výběr mezi šiframi LBlock a PRESENT, tyto šifry jsou nejvíce optimální v obou kategoriích. Na základě předchozích porovnání byla vybrána šifra LBlock z důvodu jejího menšího požadavku na logickou oblast. Dalším kritériem byla její lepší bezpečnost z pohledu známých překonání počtu kol.

5 Vývojová prostředí

V této kapitole jsou popsány vývojová prostředí Vivado Design Suit pro práci s programovací logikou (PL) a Vitis Platform pro procesní systém (PS).

5.1 Prostředí Vivado Design Suit

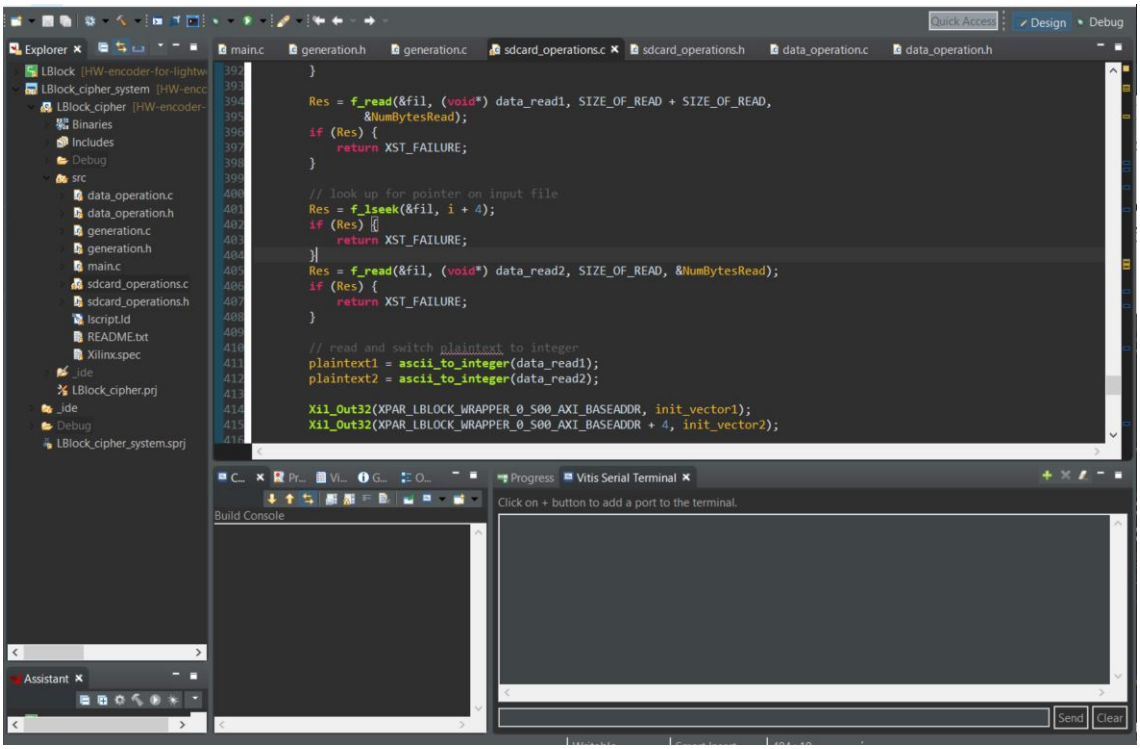
Vivado Design Suit od společnosti Xilinx Inc. je vývojové prostředí (IDE), sloužící k vývoji a implementaci návrhů pro čipy FPGA. Uvedené vývojové prostředí je zobrazeno na obrázku 5.1, který je zjednodušeně popsán v následujícím textu. V levé části se nachází panel pomocí kterého se nastavují a spouští jednotlivé kroky při vývoji. Těmito kroky jsou – návrh blokového designu, syntéza VHDL kódu, simulace modulů, implementace k ověření kódu na konkrétní FPGA a generace bitstreamu pro nahrání na FPGA čip. Levý horní panel označený jako „Sources“ obsahuje zdrojové kódy pro implementaci a simulace. Pravý panel obsahuje informace o celé implementaci a všech jejich krocích, slouží k úpravě zdrojových kódů. Spodní panel zahrnuje výpisy o aktuálních a proběhlých procesech.



Obr. 5.1: Prostředí Vivado Design Suit.

5.2 Prostředí Vitis Platform

Vitis Platform od společnosti Xilinx Inc. je upravená verze vývojového prostředí Eclipse. Slouží pro vývoj aplikací běžících na procesním systému, které se dají používat v podobě „bare metal“ nebo fungujících na linuxové distribuci PetaLinux. Podporuje jazyky C, C++, Python a jejich aplikační rámce (framework). Vývojové prostředí je zobrazeno na obrázku 5.2, který je popsán dále v textu. V levé části je průzkumník celého projektu označený „Explorer“, který obsahuje zdrojové kódy, knihovny a pomocné soubory. Pravé největší okno slouží k vytváření a úpravě kódu. Nad tímto oknem se nachází přepínání mezi rozložením pro vytváření (Design) a ladění (Debug), kdy při přepnutí do ladění se objeví navíc okno pro sledování proměnných. V pravé spodní části označené „Vitis Serial Terminal“ se nachází okno pro vstupy a výstupy konzole aplikace propojené s vývojovou deskou pomocí rozhraní UART (Universal Asynchronous Receiver-Transmitter).



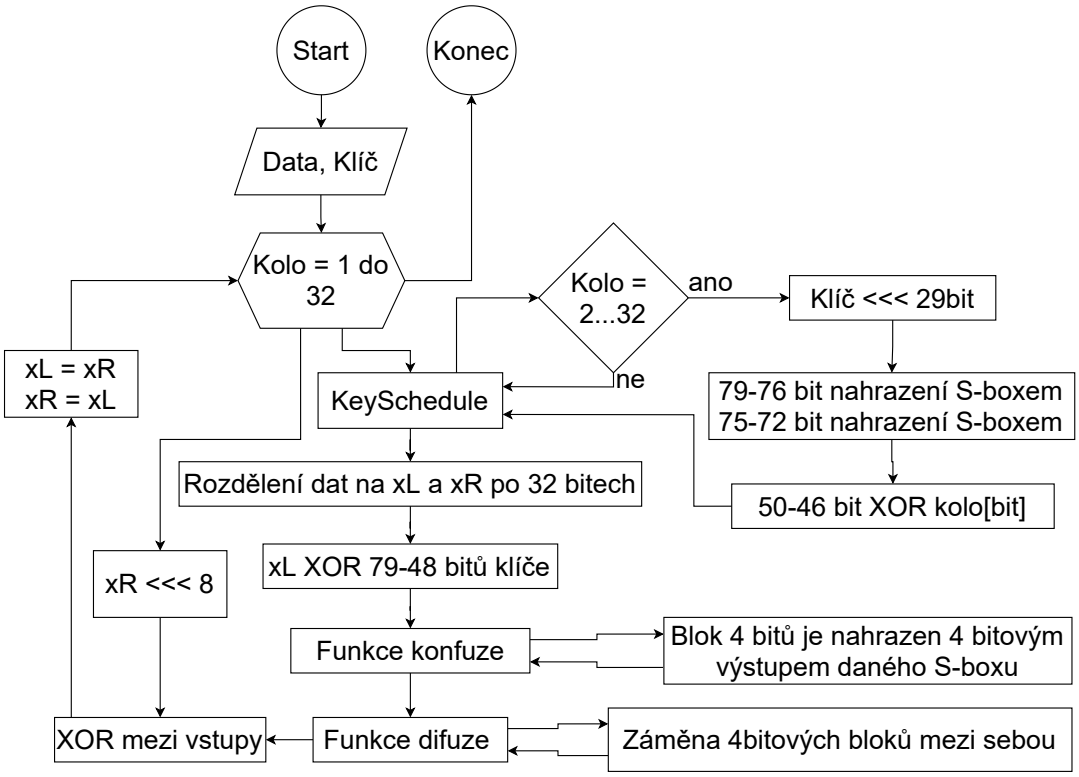
Obr. 5.2: Prostředí Vitis Platform.

6 Implementace šifry LBlock

V této kapitole se řeší přímá implementace vybrané šifry LBlock, která je detailně popsána v kapitole 3.3.2 LBlock. Algoritmus je do strojové podoby přepsán pomocí jazyka VHDL, který slouží k popsání hardwarové sekvence instrukcí. Dále je uvedeno testování implementace. Nakonec je popsán posuvný registr s lineární zpětnou vazbou.

6.1 Implentace šifry LBlock pomocí VHDL

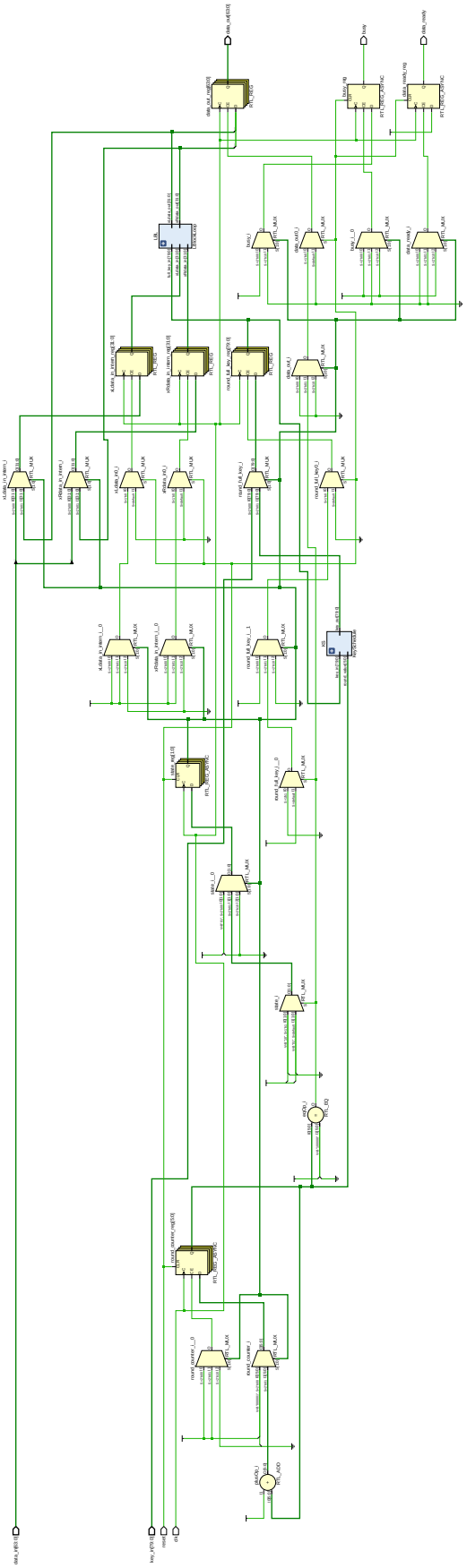
Implementace se skládá z částí, které jsou vytvořeny podle vývojového diagramu zobrazeného na obrázku 6.1. Tyto části budou podrobněji probrány v následujících podkapitolách pod názvy RoundFunction, ConfusionFunction, DiffusionFunction, KeySchedule, LBlockLoop a LBlockTop.



Obr. 6.1: Vývojový diagram šifry LBlock.

6.1.1 Zápis pomocí úrovně meziregistrových přenosů

Úroveň meziregistrových přenosů anglicky Register Transfer Level (RTL) je zápis pomocí schématu, které ukazuje propojení jednotlivých registrů a logických operací. Zápis šifry LBlock pomocí RTL je zobrazen na obrázku 6.2.



Obr. 6.2: LBlock schéma pomocí RTL.

6.1.2 Implementace KeySchedule

Implementace tohoto procesu je založena na úpravě klíče v každém kole s výjimkou prvního. V prvním kole se použije prvních 32 bitů z levé strany vstupního klíče. V následujících kolech to jsou kola 2-32. V těchto kolech probíhají operace bitového posunu jako rotace o 29 bitů. Posun je na řádku 1 ve výpisu kódu 6.1. Zde je využit vnitřní signál procesu, který slouží jako proměnná pro bitový posun klíče. Funkce S9 řádek 3 a S8 řádek 9 je využití S-boxů, které se aplikují na prvních 8 bitů rozdělených do dvou stejně dlouhých bitových bloků. Řádky 15 až 18 definují zbylých 72 bitů, kdy se s bity 50-46 a bitovou hodnotou aktuálního kola provádí operace XOR.

Výpis 6.1: Key schedule.

```
1  shifted <= key_in(50 downto 0) & key_in(79 downto 51);
2
3      S9 : Sbox9
4          port map (
5              data_in => shifted(79 downto 76),
6              data_out => key_out(79 downto 76)
7      );
8
9      S8 : Sbox8
10         port map (
11             data_in => shifted(75 downto 72),
12             data_out => key_out(75 downto 72)
13     );
14
15     key_out(71 downto 51) <= shifted(71 downto 51);
16     key_out(50 downto 46) <= shifted(50 downto 46)
17     XOR round_value(4 downto 0);
18     key_out(45 downto 0) <= shifted(45 downto 0);
```

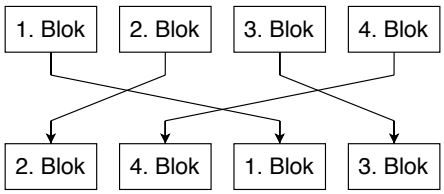
6.1.3 Implementace Sbox

Implementace modulu Sbox funguje na principu tzv. obalu neboli modul volá několik submodulů. Vstupem uvedeného modulu je blok o velikosti 32 bitů. Tento modul následně volá 8 submodulů, které implementují dané S-boxy $s_7 - s_0$. Vstupem každého S-boxu je 4bitový blok, kde do S-boxu s_7 vstupují první 4 nejvýznamnější bity a následně se k dalším S-boxům přiřazují 4bitové bloky sestupně od nejvýznamnějšího bloku. Výstupem submodulů jsou přeměněné bloky bitů, které se následně spojují

do výstupního 32bitového bloku v pořadí $s_7 - s_0$. Tento 32bitový blok je výsledným výstupem modulu Sbox.

6.1.4 Implementace DiffusionFunction

Modul DiffusionFunction je jednoduchou implementací funkce difuze. Funguje na principu vstupu 32bitového bloku dat. Blok je rozdělen na menší 4bitové bloky, které se následně vyměňují mezi sebou. Tato výměna se vždy opakuje po čtyřech 4bitových blocích. Výměna probíhá následovně tak, že první blok se přemísťuje na třetí pozici, druhý blok se přemísťuje na první pozici, třetí blok na čtvrtou pozici a čtvrtý blok na druhou pozici. Popsaný proces výměny proběhne dvakrát, to znamená, že pro každou polovinu vstupního bloku bude daný postup vykonán jednou. Po procesu výměny se všechny tyto 4bitové bloky spojí do 32bitového výstupního bloku, který je zároveň výstupem uvedeného modulu. Na obrázku 6.3 je znázorněn postup výměny čtyř 4bitových bloků.



Obr. 6.3: Výměna čtyř 4bitových bloků.

6.1.5 Implementace RoundFunction

Modul RoundFunction se stará o imlementaci funkce kola F. Vstupem do modulu je aktuální klíč kola a blok dat z levé strany Feistelovy sítě. V počáteční části se zavolá submodul, který má za úkol vyextrahovat ze vstupního 80bitového klíče prvních 32 bitů. Tento submodul je vytvořen hlavně z důvodu přehlednosti a čitelnosti kódu. Vyextrahovaný klíč se uloží do interního signálu, nesoucího název `key32bit`. Následně je vypočítán interní signál `reresultOfXOR`, který je výsledkem operace XOR mezi `key32bit` a vstupním blokem dat. Volá se submodul Sbox popsaný v 6.1.3 Implementace Sbox. Jeho vstupem je signál `resultOfXOR` a výstupem interní signál `resultOfSbox`. V poslední části se volá modul popsaný v 6.1.4 Implementace DiffusionFunction, který využívá signál `resultOFSbox` jako vstupní blok dat. Výstupní data modulu, který implementuje funkci difuze, jsou uložena do výstupu modulu RoundFunction.

6.1.6 Implementace LBlockLoop

V tomto modulu se uskutečňuje celé kolo pro šifru LBlock. Vstupem je 80bitový klíč, levá a pravá strana Feistelovy sítě. Výstupem je pravá strana, která je rovna hodnotě na vstupu levé strany a levá strana, která je výstupem úprav kola. Tyto úpravy se uvádí v postupných krocích. Nejprve se zavolá modul `RoundFunction`, který byl popsán v 6.1.5 Implementace RoundFunction. Data pravé vstupní strany, která se upraví bitovou rotací o 8 bitů, se uloží do interního signálu `shifted`. Na tento signál společně s výstupem `RoundFunction` je následně použita operace XOR. Výsledek exkluzivní disjunkce je použit na výstup levé strany. Celý postup ve VHDL je znázorněn na výpisu kódu 6.2.

Výpis 6.2: LBlockLoop.

```
1  RF : RoundFunction
2      port map (
3          key_in => fullKey_in,
4          xL_in => xLdata_in,
5          xL_out => xLhelp
6      );
7  shifted <= xRdata_in(23 downto 0) &
8             xRdata_in(31 downto 24);
9  dataHelp <= shifted XOR xLhelp;
10 xLdata_out <= dataHelp;
11 xRdata_out <= xLdata_in;
```

6.1.7 Implementace LBlockTop

V modulech obsahující slovo „top“ bývá nejčastěji uložena ovládací logika ostatních modulů. Moduly s tímto přívlastkem se používají k lepší organizaci hierarchie souborů. Tento modul implementuje časování každého kola pomocí signálu `clk` (clock). Při nástupu hrany `clk` neboli změny hodnoty z 0 na 1 se provede jedno celé kolo šifry LBlock a úpravy klíče. Signál `state` se skládá ze tří stavů, tyto stavy jsou WAITDATA, ROUND a DATAREADY. Když je signál ve stavu WAITDATA, načítá se hodnota klíče a data do interních signálů. Dále se nastaví hodnota kola a nový stav ROUND. Při stavu ROUND se nastavují hodnoty signálů z volaných modulů KeySchedule a LblockLoop, aby mohly být použity jako vstupní signály při další nástupní hraně `clk`. Dále se zde inkrementuje hodnota kola a kontroluje se zda hodnota kola nedosáhla 32. Jestliže hodnota je rovna 32, data se zapíší na výstup a hodnota stavu se změní na DATAREADY. Nastane-li hodnota menší než 32, stav se nezmění a klíč pro další kolo bude předán signálu klíče.

Výpis 6.3: LBlockTop.

```
elsif CLK'EVENT and clk = '1' then
    case state is
        when WAITDATA =>
            round_full_key <= key_in;
            xLdata_in_intern <= data_in(63 downto 32);
            xRdata_in_intern <= data_in(31 downto 0);
            busy <= '1';
            state <= ROUND;
            round_counter <= "000001";
        when ROUND =>
            xLdata_in_intern <= xLdata_out_intern;
            xRdata_in_intern <= xRdata_out_intern;
            round_counter <= round_counter + '1';
            if round_counter = "100000" then
                data_out <= xRdata_out_intern &
                xLdata_out_intern;
                state <= DATAREADY;
            else
                state <= ROUND;
                round_full_key <= round_key_help;
            end if;
        when DATAREADY =>
            busy <= '0';
            data_ready <= '1';
            state <= WAITDATA;
```

6.2 Validace implementace šifry

Testování je založeno na kontrole výstupních dat pomocí simulace ve Vivado Design Suite. Existují dvě metody zadávání vstupů do simulace. Těmito metodami je zadávání vstupů automatizovaně nebo manuálně.

Manuální zadávání vstupních signálů je vhodné, jestliže máme malý vzorek vstupů a neprovádíme testy opakovaně. V dlouhodobém časovém úseku jsou tyto testy považovány za příliš komplikované, jelikož při každém testu musíme nastavit dané hodnoty manuálně.

Automatizované testy se využívají ke kontrole více hodnot a jakmile jsou napsány, lze je opakovaně simulovat. Testy se implementují pomocí tzv. testovacího stolu

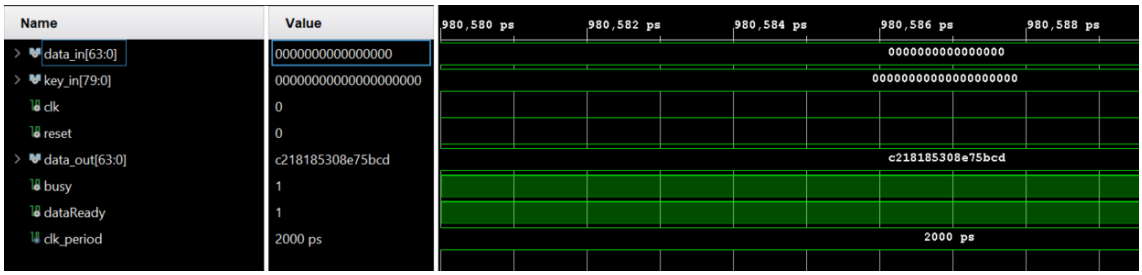
(anglicky testbench). Výsledky těchto testů mohou být kontrolovány manuálně nebo automatizovaně. V práci jsou použity pouze testy s manuální kontrolou. Výsledky simulace jsou manuálně kontrolovány s dodanými tabulkami v dokumentaci šifry (*Wenling, 2011* [30]) nebo vlastními vypočtenými hodnotami.

Za nejdůležitější lze považovat testování implementace celé šifry, která se nachází v modulu LBlockTop. Zde byly aplikovány dvě testovací hodnoty vytvořené tvůrci šifry LBlock, které byly využity pro testování. Jedná se dva vstupní bloky a dva klíče, jak je patrné z tabulky 6.1.

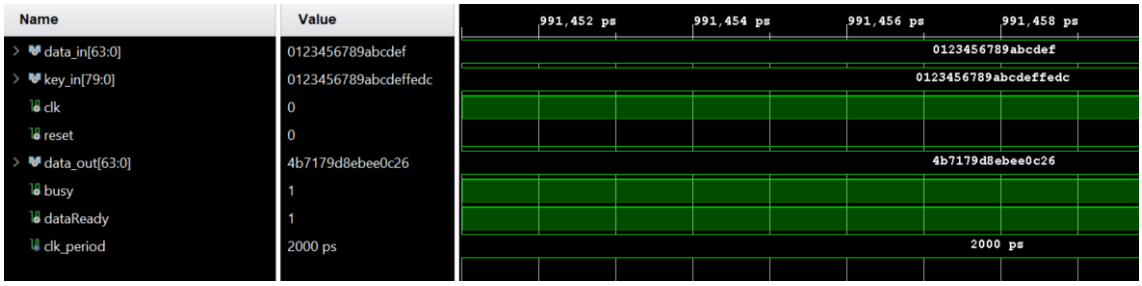
Tab. 6.1: Výsledné vektory pro LBlock v hexadecimálním tvaru [30].

	Zpráva	Klíč	Šifrovaný text
1	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00	c2 18 18 53 08 e7 5b cd
2	01 23 45 67 89 ab cd ef	01 23 45 67 89 ab cd ef fe dc	4b 71 79 d8 eb ee 0c 26

Hodnoty zprávy a klíče z tabulky 6.1 byly použity jako vstupy, kdy výsledný šifrovaný text odpovídal hodnotám dosažených v testové simulaci. Oba dva výsledky lze vidět na obrázku 6.4 pro zprávu 1 a na obrázku 6.5 pro zprávu 2.



Obr. 6.4: Výsledek simulace pro zprávu 1.



Obr. 6.5: Výsledek simulace pro zprávu 2.

6.3 Posuvný registr s lineární zpětnou vazbou

Posuvný registr s lineární zpětnou vazbou zkráceně LFSR je posuvný registr pro generování pseudonáhodných hodnot, kdy pro generaci nové hodnoty se použije jako vstup předcházející hodnota. Je tvořen kombinací bistabilních klopných obvodů a hradel typu XOR (Exclusive Disjunction) nebo XNOR (Negated Exclusive Disjunction). Počet hodnot bez opakování záleží na počtu použitých bitů. Tento počet lze získat dosazením do vzorce $2^n - 1$, kde pro n platí $n \in \mathbb{N}^+$ a zároveň určuje počet bitů [31].

Implementovaný modul pro posuvný registr s lineární zpětnou vazbou je používán pro generování 32bitových čísel, které jsou využity jako inicializační vektor a pro generování klíče. Po dosazení do výše zmíněného vzorce $2^{32} - 1$ se stejná hodnota objeví až po 4 294 967 295 cyklech. Jako vstupní hodnota pro LFSR je použito náhodné prvočíslo v bitové podobě a jeho dekadická hodnota je 3 444 898 267. V dalších cyklech je jako vstupní hodnota použita hodnota z předchozího kola. Pro získání nové hodnoty se ze vstupní hodnoty vypočte takzvaný „tap“ a vstupní hodnota se posune bitově doleva. Hodnota „tap“ se získá pomocí charakteristického polynomu, který se pro každé n liší. V případě $n = 32$ je charakteristický polynom definován jako $x^{32} + x^{22} + x^2 + x + 1$. V těchto charakteristických polynomech exponent určuje, z jakých pozic budou použity bity pro výpočet hodnoty „tap“. Hodnota „tap“ se vypočítá aplikací operace XNOR na tyto bity. Nová hodnota vznikne posunem vstupní hodnoty o jeden bit doleva a záměnou posledního bitu za hodnotu „tap“. Hodnota pro charakteristický polynom byla přebrána z článku Alfke P. (1996) [32], kde jsou uvedeny parametry charakteristického polynomu pro hodnoty n, kde $n \in \langle 1, 168 \rangle$.

Tato implementace je znázorněna ve výpisu kódu 6.4 zapsaného pomocí VHDL.

Výpis 6.4: 32bitový posuvný registr s lineární zpětnou vazbou.

```
1  begin
2    process(clk)
3    begin
4       if rising_edge(clk) then
5          current_state <= next_state;
6       end if;
7       xnor_tap <= current_state(31) xnor current_state(21)
8       xnor current_state(1) xnor current_state(0);
9       next_state <= current_state(30 downto 0) & xnor_tap;
10      random_data_out <= current_state;
11 end process;
```

7 Intellectual property blok pro šifru LBlock

V této kapitole je popsána implementace intellectual property (IP) bloku[1] pomocí AXI (Advanced eXtensible Interface). AXI rozhraní je v první části definováno z teoretického hlediska a následně je popsána implementace šifry LBlock pomocí tohoto rozhraní.

7.1 Pokročilé rozšiřitelné rozhraní

Protokol AXI je částí specifikace AMBA (Advanced Microcontroller Bus Architecture) a využívá se k propojení primárního rozhraní se sekundárním rozhraním. Společnost Xilinx Inc. převzala konkrétně protokol AXI4, který podporuje až 256 datových přenosů v jednom hodinovém cyklu na jedinou adresu. Protokol AXI4 se skládá z 5 nezávislých kanálů – *write address, write data, read address, read data* a *write response*. Každý z těchto kanálů obsahuje informační signály *VALID* a *READY*, které provádějí dvoucestný handshake[33].
AXI4 rozhraní od společnosti Xilinx Inc. se dělí na 3 typy:

- AXI4
- AXI4-Lite
- AXI4-Stream

AXI4 slouží pro paměťově mapované rozhraní a je navrženo dle specifikace AMBA. AXI4-Lite je odlehčená verze plného AXI4. Podporuje pouze jeden přenos za jeden hodinový cyklus. Je jednoduché jak z hlediska vytváření tak používání.
AXI4-Stream odstraňuje potřebu pro adresní fázi a umožňuje neomezený počet datových přenosů v jednom cyklu. Jelikož AXI4-Stream neobsahuje adresní fázi, nepovažuje se jako paměťově mapovaný. Tento typ rozhraní není jako jediný definován ve specifikaci AMBA [34].

Standardizace AXI4 umožňuje vytvářet IP bloky, které splňují určité požadavky. To vede vývojáře, aby nemuseli studovat několik typů protokolů, ale jen AXI4 protokol při vývoji IP bloků.

7.2 Implementace šifry LBlock pomocí IP bloku

K implementaci vlastního IP bloku bylo vybráno rozhraní AXI4-Lite. AXI4-Lite využívá pro komunikaci jednoho sekundárního rozhraní. Šířka pásma pro komunikaci je 32 bitů za jeden hodinový cyklus. Šířka je nastavena jako výchozí a z pohledu IP bloku ji lze považovat jako optimální. V případě použití nižší hodnoty šířky se zvýší

[1] Označován jako IP core nebo IP blok.

počet cyklů čtení a zápisu. Pokud by byla zvolena vyšší hodnota, dojde k zbytečnému zahazovaní nepotřebných bitů. Pro optimálnost a vhodné propojení s procesním systémem byly k výpočtu využity mocniny čísla 2, konkrétně $2^3, 2^4, 2^5, 2^6$. Hodnoty nadbytečných bitů a počet cyklů pro zápis 64 bitů zprávy a 80 bitů klíče pro šifru LBlock jsou uvedeny v tabulce 7.1. K těmto hodnotám se musí přičíst ještě 64 bitů pro čtení získané zprávy.

Tab. 7.1: Porovnání počtu cyklů a nadbytečných bitů pro komunikaci IP bloku.

Bitová hodnota	Nadbytečný bity	Počet cyklů zápisu	Počet cyklů čtení	Cyklů celkem
8 bitů	0 bitů	18	8	26
16 bitů	0 bitů	9	4	13
32 bitů	16 bitů	5	2	7
64 bitů	48 bitů	3	1	4

Implementace probíhala do předem připravené šablony, která řeší problémy s mapováním do paměti a automatizovanou komunikaci s procesním systémem. Šablona se skládá ze dvou částí. První část se stará o vstup a výstup signálů. Druhá část má za úkol veškerou logiku a ukládaní hodnot do registrů a jejich následné čtení. Druhá část je doplněna o vlastní logiku, skládající se z pomocných signálů a volání hlavního modulu `LBlockTop`. Pro vstupy a výstupy modulu `LBlockTop` jsou použity hodnoty registrů, které se předávají pomocným signálům, vstupujících do modulu `LBlockTop`. Všechny procesy jsou řízeny stejnými hodinami.

Výpis 7.1: LBlockTop_wrapper.

```
1  IP: LBlockTOP port map(
2            data_in => data_in_help,
3            key_in => key_help,
4            clk => S_AXI_ACLK,
5            reset => S_AXI_ARESETN,
6            data_out => data_out_help
7        );
8    data_in_help <= slv_reg0(31 downto 0)
9    & slv_reg1(31 downto 0);
10   key_help <= slv_reg2(31 downto 0)
11   & slv_reg3(31 downto 0) & slv_reg4(31 downto 16);
12   slv_reg5 <= data_out_help(63 downto 32);
13   slv_reg6 <= data_out_help(31 downto 0);
```

8 Blokový návrh

V této kapitole je popsán blokový návrh pro komunikaci mezi procesním systémem (PS) a programovací logikou (PL) a dalšími prvky na vývojové desce Zybo Z7-20. První část se věnuje komponentům v návrhu a druhá část popisuje hardwarové nároky potřebné pro tento návrh.

8.1 Popis blokového návrhu

Blokový návrh je graficky popsán na obrázku 8.1. V blokovém návrhu se nachází celkem 8 bloků a 4 I/O prvky pro propojení s vývojovou deskou. Těmito bloky jsou `ZYNQ7 processing system`, `processor system reset`, `AXI interconnect`, `LFSR32bit`, `LBlock_wrapper` a tři AXI GPIO (General Purpose Input/Output) IP bloky. Čtyřmi I/O prvky jsou `DDR`, `FIXED_IO`, `btn0 a rgb_led`.

`ZYNQ processing system` je důležitou částí celého návrhu a slouží k indikaci použití procesního systému. Na vývojové desce ZYBO Z7-20 je reprezentován procesorem typu ARM, který je zabudován přímo v FPGA čipu. Má na starost komunikaci s I/O prvky a periferiemi na vývojové desce. Tyto vstupy a výstupy můžou být například – UART (Universal Asynchronous Receiver-Transmitter), HDMI (High-Definition Multimedia Interface), ethernet, USB (Universal Serial Bus) porty, slot na microSD, různé typy pamětí a jiné. Komunikovat s těmito vstupy a výstupy se může pomocí takzvaného „bare metal“ nebo některé z linuxových distribucí, kdy nejčastější distribucí je PetaLinux. Uvedenými metodami komunikace jsou následně vytvářeny aplikace, které urychlují své výpočty prostřednictvím navržené logiky na PL. Procesní systém se také stará o generaci signálu přerušení a systémových hodin pro celý návrh.

`Processor system reset` je IP blok, který obsluhuje požadavky přerušení tak, aby byly jednotné pro celý návrh. Obsahuje několik typů vstupů pro jednotlivá přerušení a zajišťuje odpovídající přerušení pro rozdílné typy výstupů.

`AXI interconnect` je jedním z důležitých IP bloků. Má za úkol propojit několik primárních a sekundárních AXI rozhraní, které využívají mapování paměti. Tento IP blok propojuje jedno primární rozhraní se čtyřmi sekundárními. Primárním AXI rozhraním je rozhraní pro obecné použití umístěné na procesním systému ZYNQ7 popsaném výše a označeném jako `M_AXI_GP0`. Sekundárními rozhraními jsou tři AXI GPIO IP bloky a navržený IP blok `LBlock_wrapper`. Tyto sekundární rozhraní jsou vysvětleny dále.

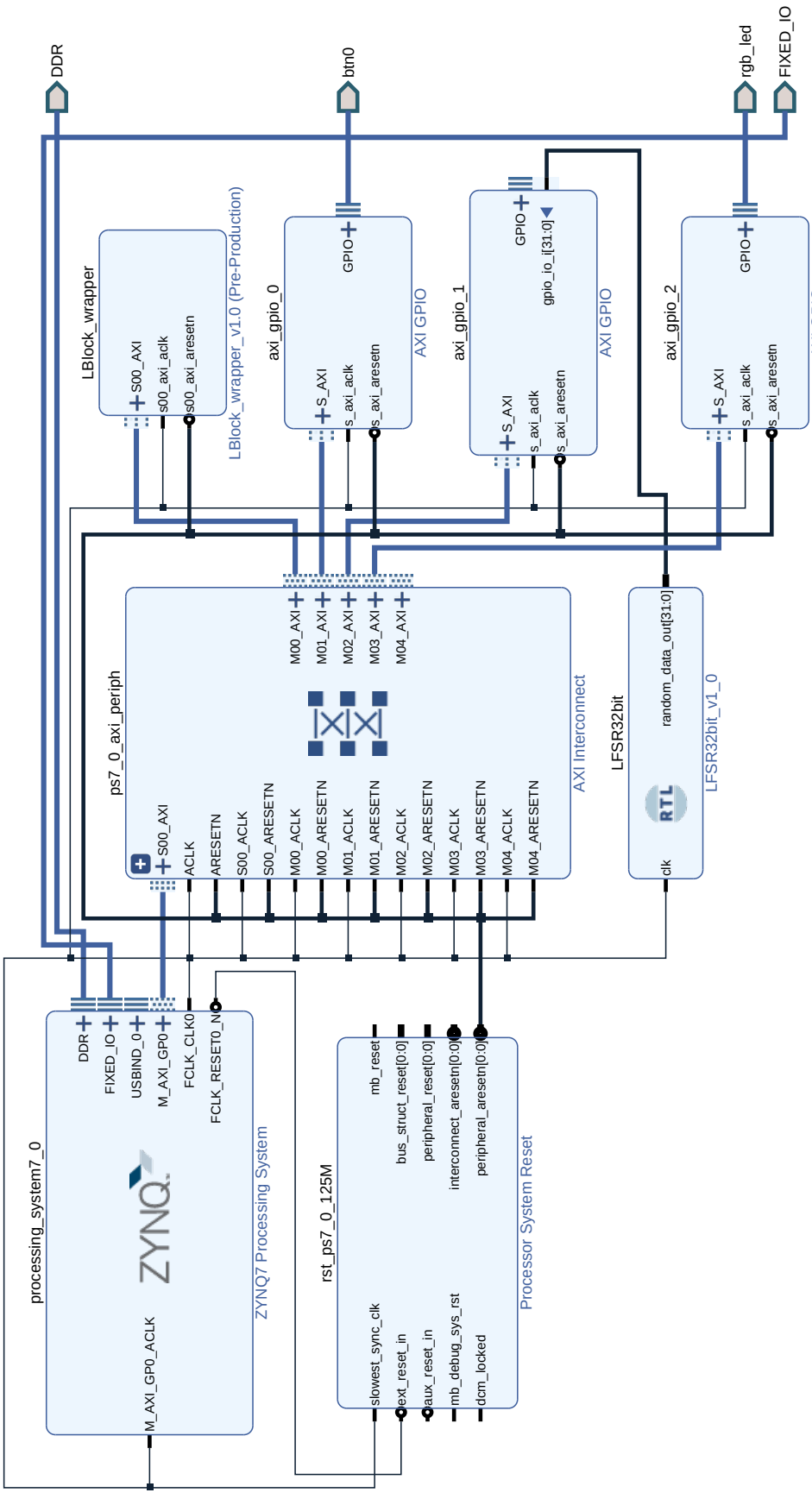
`LFSR32bit` je 32bitový generátor pseudonáhodných čísel, který je vložen v podobě RTL[1] bloku a popis jeho funkčnosti je uveden v kapitole 6.3 Posuvný registr s lineární zpětnou vazbou. Výstup bloku je směřován do jednoho ze tří AXI GPIO IP bloku.

`LBlock_wrapper` je vlastní IP blok, který se stará o šifrování dat poslaných z procesního systému a vrácení těchto dat zpátky procesnímu systému. Tento IP blok je detailněji popsán v kapitole 7.2 Implementace šifry LBlock pomocí IP bloku.

GPIO IP blok slouží k propojení I/O rozhraní s AXI rozhraním. S ostatními AXI rozhraními, které jsou jako primární rozhraní, komunikuje pomocí sekundárního AXI rozhraní. S bloky a prvky, které nemají definované AXI rozhraní, umožňuje vytvořit komunikaci. GPIO IP blok poskytuje možnost komunikace pomocí jednoho nebo dvou rozhraní GPIO. Každému GPIO rozhraní lze nastavit šířku přenášených dat mezi 1 bitem až 32 bity a uvést, jestli se jedná o vstupní, výstupní nebo vstupně-výstupní kanál. V návrhu jsou použity dva vstupní a jeden výstupní GPIO IP bloky. První vstupní blok je označen `axi_gpio_0`, jeho šířka přenosu je nastavena na 4 bity a slouží pro komunikaci se čtyřmi hlavními tlačítky pro stisknutí na vývojové desce. Druhým vstupním blokem je blok označený `axi_gpio_1` s 32bitovou šířkou přenosu. Tento blok má na starost přijímat výstupy z generátoru pseudonáhodných čísel. Jediným výstupním blokem o šířce přenosu 5 bitů je blok s označením `axi_gpio_2`, umožňujícím ovládaní RGB LED umístěnými na vývojové desce.

Dva ze čtyř I/O prvků jsou připojeny na GPIO IP blok. Těmito prvky jsou `btn0` a `rgb_led`. Prvek `btn0` se využívá pro propojení se čtyřmi fyzickými tlačítky, označené BTN0, BTN1, BTN2 a BTN3. Prvek `rgb_led` slouží k propojení se dvěma RGB LED, označené na desce jako LD5 a LD6. Piny pro tyto tlačítka a RGB LED jsou přesně definovány výrobcem a jsou alokovány v I/O bankách FPGA. Alokované I/O banky pro tlačítka jsou K18, P16, K19, Y16 a pro RGB LED to jsou Y11, T5, Y12, V16, F17, M17. Zbylé dva I/O prvky jsou požadovány pro PS.

[1] Úroveň meziregistrových přenosů z anglického Register Transfer Level



Obr. 8.1: Návrh šifrátoru.

8.2 Hardwarové nároky na blokový návrh

8.2.1 Hardwarové nároky celého návrhu

Celkový návrh po implementaci je velmi nenáročný. Odhadované potřebné zdroje při implementaci by měli mít velikost 976 LUT a 1583 flip-flop. Z těchto 976 LUT je určeno 62 pro paměť, konkrétně pro registry. Ve 420 takzvaných „řezech“ (slice), dříve označovaných jako logické buňky, jsou uloženy LUT a flip-flop. Spojením dvou řezů získáme jeden konfigurovatelný logický blok. V tabulce 8.1 jsou rozepsány tyto a další zdroje s jejich maximálním počtem na vývojové desce a procentuální využití zdrojů.

Tab. 8.1: Odhadované potřebné zdroje návrhu.

Název zdroje	Využité zdroje	Maximální zdroje	Procentuální využití
LUT celkem	976	53 200	1,83%
LUT jako logika	914	53 200	1,72%
LUT jako paměť	62	17 400	0,36%
Flip-flop	1583	106 400	1,49%
Řez (slice)	420	13 300	3,16%
F7 multiplexory	32	26 600	0,12%
Bonded I/O bloky	10	125	8%
Bonded I/OPADs	130	130	100%

Požadavky na jednotlivá primitiva[2] jsou zobrazeny v tabulce 8.2. Hlavním rozdílem oproti konečným potřebným zdrojům je větší počet LUT. Důvodem rozdílu je optimalizační technika při implementaci ve Vivado Design Suite. Tato technika má na starost optimalizaci cest a proto spojuje menší LUT do větších. Správnost výpočtu lze ověřit pomocí příkazů ve výpisu 8.1, kde tyto příkazy byly převzaty z oficiálního fóra Xilinx. Z příkazů lze zjistit že bylo zkombinováno 229 LUT.

V tabulce 8.2 jsou uvedeny konkrétní typy flip-flop. FDRE je flip-flop typu D s hodinami reagujícími na nástupní hranu a synchronním signálem *reset*. FDSE je flip-flop typu D s hodinami reagujícími na nástupní hranu a synchronním signálem *set*. FDCE je flip-flop typu D s hodinami reagujícími na nástupní hranu a asynchronním signálem *clear*. FDPE je flip-flop typu D s hodinami reagujícími na nástupní hranu a asynchronním signálem *preset*. OBUF je označení pro výstupní vyrovnávací paměť, v případě použitého návrhu slouží pro RGB LED. IBUF je označení pro

[2]Typy zdrojů jako je LUT, flip-flop, I/O a jiné.

vstupní vyrovnávací paměť, sloužící ve stejném případě pro tlačítka. BIBUF je obousměrný vyrovnávací paměť, který slouží pevně daným I/O vstupům pro propojení s procesním systémem označeným v tabulce 8.1 jako I/OPADs.

Tab. 8.2: Požadavky na jednotlivá primitiva.

Primitivum	LUT6	LUT5	LUT4	LUT3	LUT2	LUT1	LUT celkově
Počet	326	229	143	388	105	17	1208
Primitivum	FDRE	FDSE	FDCE	FDPE	FF celkově		MUXF7
Počet	1513	62	7	1	1583		32
Primitivum	OBUF	IBUF	BIBUF		IO celkově		
Počet	6	4	130		140		

Výpis 8.1: Skript pro zobrazení počtu sloučených LUT.

```
1  set total_luts [llength [get_cells
2  -hierarchical -filter { PRIMITIVE_TYPE =~ LUT.*.LUT* } ]]
3
4  set combined_luts [llength [get_cells -hierarchical
5  -filter { PRIMITIVE_TYPE =~ LUT.*.LUT5 }]]
6
7  set final_logic_lut_count
8  [expr $total_luts - $combined_luts]
9
10 puts "Slice LUTs: $final_logic_lut_count"
```

8.2.2 Hardwarové nároky jednotlivých částí návrhu

V tabulce 8.3 jsou zobrazeny hardwarové nároky pro jednotlivé bloky. Tyto hodnoty odpovídají hodnotám primitivů, protože u nich nebyla provedena optimalizace.

Procesní systém nevyužívá žádné LUT a flip-flops, jelikož je to procesor typu ARM a s programovací logikou pouze komunikuje pomocí 130 vstupů a výstupů. Nejvíce náročný je `AXI interconnect` blok s využitím 665 LUT a 650 flip-flop. Nejméně náročný je blok pro generaci pseudonáhodných čísel `LFSR32bit` využívajících 1 LUT a 32 flip-flop. IP blok pro šifru LBlock využívá 327 LUT a 420 flip-flop, kdy samostatný modul šifry LBlock využívá 261 LUT a 216 flip-flops. Závěrem lze vyvodit, že pro operace v IP bloku je využito 66 LUT a 204 flip-flop. V tabulce 8.4 jsou tyto závislosti rozepsány.

Tab. 8.3: Odhadované potřebné zdroje bloků návrhu.

Název bloku	LUT	Flip-flop	I/OPADs
ZYNQ processing system	-	-	130
Processor system reset	20	33	-
AXI interconnect	665	650	-
LFSR32bit	1	32	-
LBlock_wrapper	327	420	-
axi_gpio_0	48	68	-
axi_gpio_1	99	318	-
axi_gpio_2	48	62	-
Celkem	1208	1583	130

Tab. 8.4: Odhadované potřebné zdroje IP bloku pro šifru LBlock.

Název bloku	LUT	Flip-flop	MUX
Modul LBlockTOP	261	216	32
Logika v *LBlock_wrapper*	66	204	0
Celý blok *LBlock_wrapper*	327	420	32

Pro porovnání je zde uvedena tabulka 8.5 obsahující pracovní frekvenci a hodnotu LUT vybraných šifer lehké kryptografie a implementované šifry LBlock. Číslo za šifrou udává, jakou délku klíče šifra využívá. Pracovní frekvence pro šifru LBlock je nižší než u ostatních šifer, jelikož její hodnoty byly vzaty přímo z návrhu a ne z její samostatné implementace na FPGA.

Tab. 8.5: Požadované LUT pro šifry lehké kryptografie [35].

Název šifry	LBlock	PRESENT-80	LED-80	SIMON 96/96	AES-128
Počet LUT	261	311	358	435	318
Frekvence [MHz]	125	542	423	473	287

9 Implementace na procesní systém

V této kapitole je vysvětleno propojení s procesním systémem (PS), s prvky blokového návrhu a implementace šifrátoru. Poté jsou popsány jednotlivé typy implementace. Tyto typy jsou následně rozebrány a pro šifrování souborů na microSD kartě je provedeno měření rychlosti šifrování. Nakonec je popsána validace výstupu šifry.

9.1 Propojení procesního systému s prvky návrhu

Na procesním systému může fungovat některá z linuxových distribucí nebo přímo „embedded“ aplikace, která komunikuje pomocí „bare metal“. Tento šifrátor je vytvořen jako „embedded“ aplikace, aby byla možnost přenositelnosti na jakoukoliv vývojovou desku ZYBO Z7, aniž by byl potřebný operační systém.

Pro komunikaci pomocí AXI GPIO bloků se využívá základní knihovny *xgpio.h* od společnosti Xilinx. V předchozí kapitole bylo uvedeno, že s pomocí AXI GPIO komunikují tlačítka, RGB LED a LFSR blok. Tyto spojení se musí prvně inicializovat pomocí funkce `XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId)`. Prvním parametrem je předávání adresy na proměnnou typu XGpio a druhým parametrem je adresa instance AXI GPIO bloku. Pro komunikaci se poté navazují výchozí hodnoty pomocí funkce `XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel, u32 DirectionMask)`. První parametr je adresa na už inicializovanou proměnnou, druhý parametr určuje na jakém kanálu AXI GPIO se bude komunikovat a třetí parametr je bitová maska, která rozděluje vstupy a výstupy. Pro výstupy se využije nastavení bitů na 0 a pro vstupy nastavení bitů na 1.

9.1.1 Komunikace s tlačítky

Komunikace s tlačítky probíhá pomocí standardní funkce `XGpio_DiscreteRead (XGpio *InstancePtr, unsigned Channel)`, která je funkcí výstupní. První parametr je adresa na inicializovanou proměnnou, druhý parametr určuje na jakém kanálu AXI GPIO se bude komunikovat. Tato funkce je volána v nekonečném cyklu, kdy se čeká na stisk jednoho z tlačítek. Pro zjištění, které tlačítko má danou hodnotu, je uvedena tabulka 9.1. Při stisknutí se provede jeden typ šifrování popsaný dále v této kapitole 9 Implementace na procesní systém.

Tab. 9.1: Bitová hodnota tlačítek.

Název tlačítka	BTN0	BTN1	BTN2	BTN3
Bitová hodnota	0b0001	0b0010	0b0100	0b1000

9.1.2 Komunikace s RGB LED

Předání hodnot pro RGB LED probíhá pomocí funkce `XGpio_DiscreteWrite(XGpio *InstancePtr, unsigned Channel, u32 Data)`, která je funkcí vstupní. První parametr je adresa na inicializovanou proměnnou, druhý parametr určuje na jakém kanálu AXI GPIO se bude komunikovat. Třetím je hodnota, která má být předána AXI GPIO bloku. Předávána je hexadecimální hodnota, která určuje barvu diod. Jelikož je třeba rozsvítit konkrétní diodu v určité barvě, byly nalezeny hodnoty pro jednotlivé barvy – modrou, zelenou, chladně bílou, červenou, růžovou, žlutou, teple bílou. Hexadecimální hodnoty pro každou barvu jsou uvedeny v tabulce 9.2

Tab. 9.2: Hexadecimální hodnota pro RGB LED.

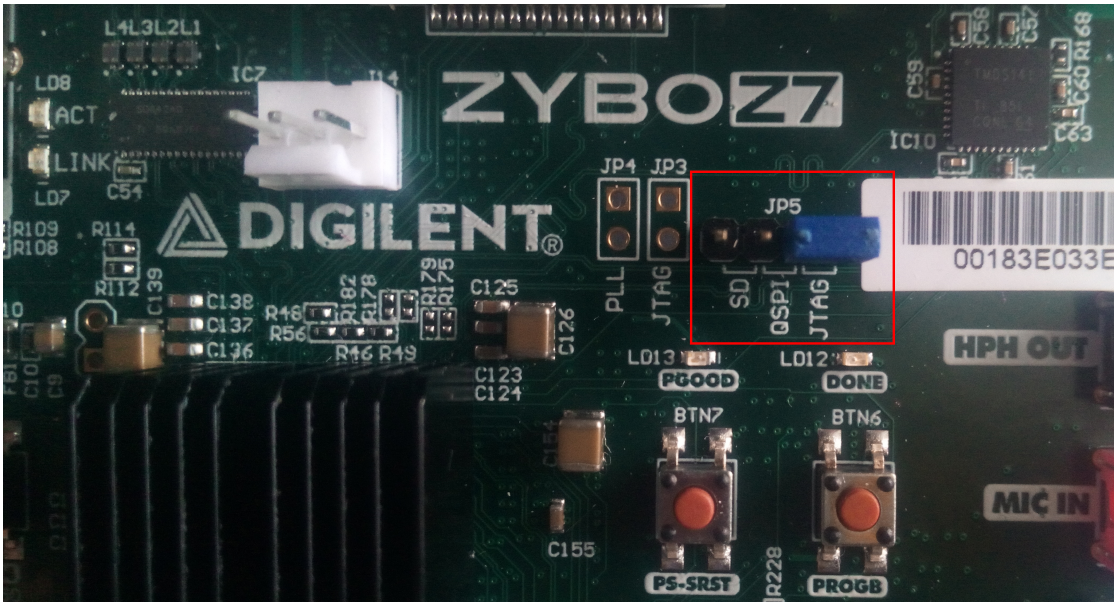
Barva	Hodnota pro levou diodu LD6	Hodnota pro pravou diodu LD5
Modrá	0x08	0x01
Zelená	0x10	0x02
Chladná bílá	0x18	0x03
Červená	0x20	0x04
Růžová	0x28	0x05
Žlutá	0x30	0x06
Teplá bílá	0x38	0x07
Žádná barva	0x00	0x00

9.1.3 Komunikace s LFSR

Komunikace probíhá podobným způsobem jako v případě tlačítek pomocí funkce `XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel)`. Hlavním rozdílem oproti výstupu z této funkce je její hodnota o délce 32 bitů, zatímco u tlačítek je délka 4 bity. Stejně jako při komunikaci s tlačítky se jedná o výstupní funkci. Hodnota výstupu slouží jako inicializační vektor (IV) pro šifrovou zpětnou vazbu šifry LBlock. Dále je také využita pro generaci náhodného klíče pro šifrování, pokud uživatel nedodá vlastní klíč. Popis generování této pseudonáhodné hodnoty je rozveden v podkapitole 6.3 Posuvný registr s lineární zpětnou vazbou.

9.2 Implementace šifrování pomocí microSD karty

Pro šifrování dat bylo zvoleno jako primární způsob šifrování pomocí microSD karty. K operacím s microSD kartou je použita knihovna FatFs, která podporuje disky formátované na souborový systém FAT (File Allocation Table). Pro úspěšnou komunikaci s microSD kartou musí být propojeny piny QSP1 a JTAG, propojení je zobrazeno na obrázku 9.1.



Obr. 9.1: Propojení pinů pro podporu microSD karty.

K testování byla využita microSD karta SanDisk Extreme 32GB a její parametry jsou uvedeny v tabulce 9.3.

Tab. 9.3: Parametry microSD karty.

Velikost [GB]	Rychlostní třída	Rychlost čtení [MB/s]	Rychlost zápisu [MB/s]
32	Class 10	100	60

Pro šifrování je nutné mít vstupní soubor, který má být šifrován v jakémkoliv formátu. Tento soubor musí nést název *input*, kde na příponě nezáleží, jelikož bude automaticky zjištěna. Druhým souborem, který není povinný, je soubor obsahující klíč s názvem *key.txt*. Textový soubor byl zvolen hlavně kvůli jednoduchosti pro uživatele. Šifrovat lze třemi způsoby, ke kterým se váží tři tlačítka – BTN1, BTN2 a BTN3. Tlačítko BTN1 slouží k šifrování pomocí výstupní zpětné vazby (OFB) s automatickou generací klíče. Tlačítko BTN2 se používá k šifrování pomocí šifrové

zpětné vazby (CFB), pokud si uživatel chce nechat vygenerovat klíč. Tlačítko BTN3 slouží také k šifrování pomocí CFB, ale využívá uživatelem dodaný klíč.

Výstupem jsou 2 respektive 3 soubory. Všechny tři způsoby mají jako výstup šifrovaný soubor a inicializační vektor, kdy se při generaci klíče k těmto dvou souborům přidá ještě soubor s klíčem. Šifrovaný soubor má název *OUTPUT.bin*, binární soubor byl vybrán z důvodu jeho možnosti otevření na všech operačních systémech. Vzhledem k přenositelnosti by se dal využít i textový soubor, který pro tuto činnost není příliš vhodný, jelikož slouží k přenosu textových dat pro člověka v čitelné podobě. V tomto případě není potřebné, aby člověk přečetl data, jelikož jsou v šifrované podobě. Do binárních souborů se mohou ukládat šifrovaná a kompresovaná data, obrázky, videa a cokoliv dalšího. Inicializační vektor je také uložen s příponou .bin a názvem *INVECTOR.bin*. Třetím výstupním souborem může být *KEY.txt*, který obsahuje klíč, zadaný přímo do souboru v binární podobě. Může to způsobit, že část textu nebude viditelná z důvodu zobrazení ASCII (American Standard Code for Information Interchange) znaků, které nemají textovou podobu. Při otevření souboru v binární formě lze zjistit, že tyto znaky jsou v souboru skutečně zapsané.

9.2.1 Průběh šifrování

Šifrování probíhá dvěma způsoby, záleží jestli uživatel volí vlastní klíč nebo je klíč generován šifrátorem. Pokud je klíč generován šifrátorem, k postupu přibývá krok generace klíče.

Generace funguje voláním LFSR, které je voláno třikrát po sobě k získaní pseudonáhodného 32bitového čísla bez znaménka. Výsledná tři čísla jsou následně převedena na 3 pole ASCII znaků. Pole znaků jsou poté uloženy do souboru, kdy z posledního pole je použita pouze polovina znaků. To je způsobeno tím, že pro klíč je potřeba 10 znaků po 8bitech.

Pro šifrování jsou použity módy výstupní zpětné vazby (OFB) a šifrové zpětné vazby (CFB), která pracuje s výstupem pro šifrování o velikosti 32 bitů. Šifrování je tvořeno z 5 fází:

1. Načtení klíče ze souboru.
2. Prohledání microSD karty pro vstupní soubor.
3. Vygenerování IV a jeho uložení do souboru.
4. Šifrování souboru.
5. Zobrazení úspěšnosti pomocí RGB LED.

V první fázi se načítá klíč ze souboru určeného pro klíč. Tyto hodnoty se následně předávají do IP bloku šifry LBlock pomocí funkce `Xil_Out32(UINTPTR Addr, u32 Value)`, kde prvním parametrem je adresa registru pro IP blok šifry LBlock

a druhým parametrem je 32bitové číslo bez znaménka. V IP bloku jsou pro klíč rezervovány 3 registry.

Ve druhé fázi se prohledává microSD karta. Při nalezení prvního souboru s názvem *input* se vrátí adresa tohoto souboru, který bude následně šifrován. Pokud by na microSD kartě bylo více souborů s názvem *input*, použije se pouze první nalezený a další soubory nebudou šifrovány.

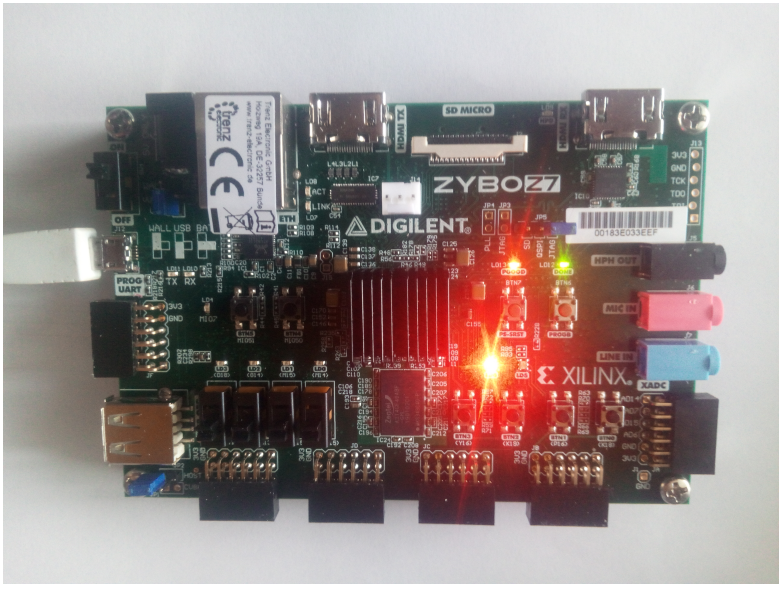
Ve třetí fázi se generuje IV. Proces probíhá stejným způsobem jako v případě generace klíče. Tento vygenerovaný IV je uložen do souboru a je předán do IP bloku jako vstupní zpráva šifry. V IP bloku jsou pro zprávu vyčleněny 2 registry.

Ve čtvrté fázi po načtení klíče a IV probíhá šifrování pomocí 32bitové CFB nebo OFB. Jelikož inicializační vektor byl načten v předchozí fázi, jako další se čte již šifrovaná hodnota IV. Uvedená hodnota se čte pomocí funkce `Xil_In32(UINTPTR Addr)`, kde jediným parametrem je adresa registru pro IP blok šifry LBlock a výstupem je 32bitové číslo bez znaménka. Číslo se čte ze dvou 32bitových registrů.

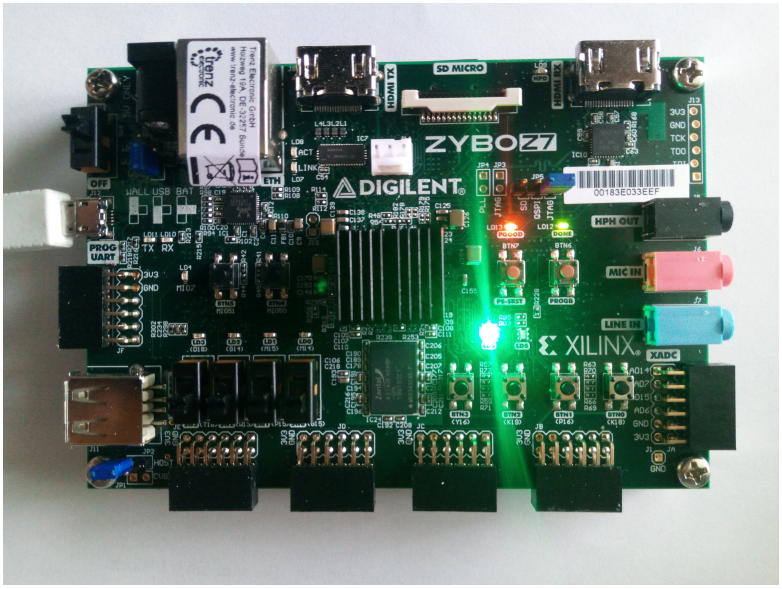
Pro CFB je dále načteno 32 bitů ze vstupního souboru ve formě pole o čtyřech 8bitových znacích. Pole je následně převedeno na 32bitové číslo bez znaménka. Na čísla je poté aplikována operace XOR pro 32 nejvíce signifikantních bitů. Výsledek této operace se převede zpět do pole znaků, kdy je toto pole uloženo do výstupního souboru. Vstupem pro další blok je 32 nejméně signifikantních bitů šifrovaného IV spojených s výsledkem operace XOR. Tento postup se opakuje do té doby, než je zašifrován celý soubor. V případě, že velikost souboru nelze dělit 4 beze zbytku, je tento postup aplikován ještě jednou. S tím rozdílem, že je zapsáno pouze tolik bytů kolik je zbytek.

Pro OFB je šifrování velmi podobné. Ze vstupního souboru se načítá 64 bitů do dvou polí o velikosti čtyř 8bitových znaků. Na šifrovaný IV a pole znaků ze vstupního souboru převedená na čísla se aplikuje operace XOR. Výsledek této operace je převeden na pole znaků a uložen do výstupního souboru. Dále je výsledek ve formě bez znaménka použit jako vstupní hodnota do dalšího kola šifrování. Operace převodu z pole znaků na číslo bez znaménka a opačně je řešena co nejefektivněji pomocí bitových posunů.

V páté fázi je rozsvícena korespondující dioda. V případě chyby v průběhu šifrování se rozsvítí dioda červeně a při úspěšném šifrování zeleně. Příklad neúspěšného šifrování je zobrazen na obrázku 9.2, úspěšného na obrázku 9.3.



Obr. 9.2: Neúspěšné šifrování na microSD kartu.



Obr. 9.3: Úspěšné šifrování na microSD kartu.

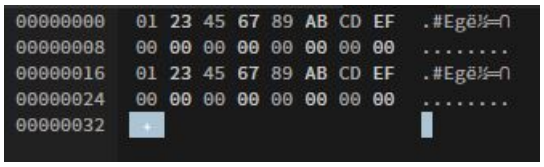
9.2.2 Testování zápisu na microSD kartu

Testování probíhalo formou zápisu do souboru, kdy byl vstupní soubor šifrován pomocí elektronické kódové knihy (ECB). Jako vstupní hodnoty byly vybrány stejné hodnoty jako při testování modulu `LBlockTOP` v tabulce 6.1, pro přehlednost znovu uvedené v tabulce 9.4.

Tab. 9.4: Vektory pro LBlock v hexadecimálním tvaru [30].

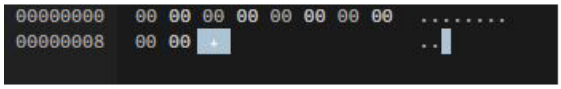
	Zpráva	Klíč	Šifrovaný text
1	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00	c2 18 18 53 08 e7 5b cd
2	01 23 45 67 89 ab cd ef	01 23 45 67 89 ab cd ef fe dc	4b 71 79 d8 eb ee 0c 26

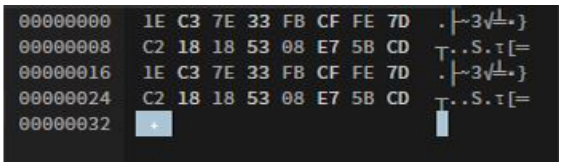
Na obrázku 9.4 lze nalézt vstupní soubor, který je zobrazený v hexadecimálním editoru. Soubor obsahuje obě vstupní zprávy.



Obr. 9.4: Vstupní soubor pro testování šifrování.

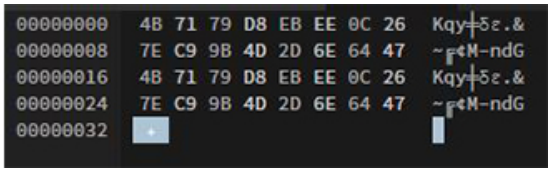
Na obrázku 9.5 lze na levé straně vidět použitý klíč pro šifrování zprávy a na pravé straně výsledný zašifrovaný soubor. V zašifrovaném souboru by dle klíče a vstupního souboru měl odpovídat druhý a čtvrtý řádek. Při porovnání s tabulkou 9.4 lze zjistit, že šifrování proběhlo v pořádku, jelikož tyto řádky odpovídají šifrované zprávě na řádku 1 v tabulce.





Obr. 9.5: Klíč a výstupní soubor pro testování šifrování pro nulové hodnoty.

Obrázek 9.6 je podobný předchozímu testování s rozdílem v hodnotě klíče. V šifrovaném souboru by měly odpovídat tabulkové hodnotě řádky jedna a tři. Jelikož i v tomto případě řádky odpovídají druhému řádku v tabulce 9.4, lze správnost implementace prohlásit za úspěšnou.



Obr. 9.6: Klíč a výstupní soubor pro testování šifrování pro nenulové hodnoty.

9.2.3 Měření rychlosti šifrování

Pro měření byly použity soubory o velikosti 1MiB, 10MiB, 50MiB a 100MiB. Obsah těchto souborů byl náhodný a klíč pro šifrování byl stejný. Pro zjištění rychlosti bylo použito kódu ve výpisu kódu 9.1. Tento kód určí, kolik cyklů a jak dlouho trvalo šifrování. Testy rychlosti byly pro každou velikost provedeny pětkrát, aby bylo zamezeno odchylce.

Výpis 9.1: Zjištění času průběhu šifrování.

```
1 XTime tStart, tEnd;
2 XTime_GetTime(&tStart);
3 // kód pro měření času
4 XTime_GetTime(&tEnd);
5 printf("Output took %llu clock cycles.\n",
6     2*(tEnd - tStart));
7 printf("Output took %.2f us.\n",
8   1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000000));
```

V tabulce 9.5 lze vidět testování rychlosti pro 32bitové CFB. Časy jsou uvedené v mikrosekundách (µs) z důvodu co největší přesnosti. V tabulce 9.6 jsou uvedeny hodinové cykly odpovídající naměřeným časům. V posledním řádku je u obou tabulek průměr ze všech měření.

Tab. 9.5: Měření rychlosti CFB šifrování v závislosti na čase.

Velikost	Čas na 1MiB [µs]	Čas na 10MiB [µs]	Čas na 50MiB [µs]	Čas na 100MiB [µs]
1 měření	3 998 393,69	37 448 793,74	187 897 558,75	375 510 117,74
2 měření	3 752 616,41	37 151 587,79	188 297 507,76	376 323 832,30
3 měření	3 777 467,65	37 305 797,15	187 260 973,63	375 041 352,88
4 měření	3 832 692,49	37 590 809,20	187 295 884,42	375 437 714,77
5 měření	3 754 944,57	37 257 961,66	186 681 240,19	380 506 015,96
Průměr	3 823 222,96	37 350 989,91	187 486 632,95	376 563 806,73

Z tabulky 9.5 lze vyčíst, že šifrování 1 MiB trvá v průměru 3 823 222,96 µs, což je přibližně 3,82 sekund. Pokud bychom zkombinovali veškeré průměry a převedli je pro časy na 1MiB, zjistíme, že šifrování v průměru trvá 3 768 423,17 µs neboli přibližně 3,77 sekund. Tato závislost je uvedena v tabulce 9.7.

Bylo také provedeno měření jednotlivých kroků, kdy samostatné šifrování 4 bytů trvá přibližně 1,08 µs a čtení ze souboru a zápis do souboru trvá pro každou operaci přibližně 0,83 µs. Lze tak vyvodit, že šifrování se čtením ze souboru a zápisem do

souboru 4bytové hodnoty trvá přibližně 2,74 µs. Z toho čtení a zápis trvá přibližně 1,66 µs, což odpovídá necelým 60,59%. Propustnost lze vypočítat jako $\frac{velikost\,[KiB]}{čas\,[s]}$, po dosazení vypočtených hodnot $\frac{1024}{3{,}76842317}$ je propustnost pomocí CFB přibližně 271,73 KiB/s.

Tab. 9.6: Počet cyklů CFB na velikost souboru.

Velikost	Cyklů na 1MiB	Cyklů na 10MiB	Cyklů na 50MiB	Cyklů na 100MiB
1 měření	2 662 930 200	24 940 896 630	125 139 774 130	250 089 738 418
2 měření	2 499 242 528	24 742 957 466	125 406 140 166	250 631 672 312
3 měření	2 515 793 454	24 845 660 904	124 715 808 436	249 777 541 018
4 měření	2 552 573 198	25 035 478 924	124 739 059 024	250 041 518 040
5 měření	2 500 793 084	24 813 802 468	124 329 705 966	253 417 006 628
Průměr	2 546 266 493	24 875 759 279	124 866 097 545	250 791 495 284

Tab. 9.7: Měření rychlosti CFB šifrování pro 1MiB v závislosti na čase.

Velikost	Čas na 1MiB [µs]	Čas na 10MiB [µs]	Čas na 50MiB [µs]	Čas na 100MiB [µs]
Průměr	3 823 222,96	37 350 989,91	187 486 632,95	376 563 806,73
Průměr na 1MiB	3 823 222,96	3 735 098,99	3 749 732,66	3 765 638,07

Výsledky testování pro OFB lze vidět v tabulce 9.8. Jelikož OFB využívá dvakrát větší bloky při zápisu, její rychlost šifrování 1MiB by měla být větší než v případě 32bitového CFB. Stejná závislost by se měla projevit také v tabulce 9.9. Pro OFB byly stejně jako u CFB využity soubory s náhodným obsahem o stejných velikostech. Obsah těchto souborů nebyl shodný s obsahem souborů použitých při testování CFB. Klíč pro šifrování byl použit stejný jako v případě CFB.

Z tabulky 9.8 lze zjistit, že šifrování 1MiB trvalo 3 398 183,82 µs, což je přibližně 3,4 sekund. Při převedení času pro šifrování větších souborů na µs za 1MiB a zprůměrováním těchto hodnot vyplynulo, že průměr je 3 516 333,61 µs, což je přibližně 3,52 sekund. Průměrné hodnoty na 1MiB lze nalézt v tabulce 9.10. Podobně jako v případě CFB lze vypočítat propustnost jako $\frac{velikost\,[KiB]}{čas\,[s]}$, po dosazení vypočtených hodnot $\frac{1024}{3{,}51633361}$ je propustnost pomocí OFB přibližně 291,21 KiB/s.

Tab. 9.8: Měření rychlosti OFB šifrování v závislosti na čase.

Velikost	Čas na 1MiB [µs]	Čas na 10MiB [µs]	Čas na 50MiB [µs]	Čas na 100MiB [µs]
1 měření	3 344 037,35	34 939 189,19	178 707 351,89	356 134 948,48
2 měření	3 431 026,56	35 558 079,39	178 393 276,71	356 681 691,40
3 měření	3 392 589,52	35 279 611,45	178 464 345,17	355 351 306,76
4 měření	3 410 118,67	35 293 686,13	178 305 986,76	356 916 313,03
5 měření	3 413 147,01	35 721 507,26	177 820 381,47	357 187 644,66
Průměr	3 398 183,82	35 358 414,68	178 338 268,40	356 454 380,87

Tab. 9.9: Počet cyklů OFB na velikost souboru.

Velikost	Cyklů na 1MiB	Cyklů na 10MiB	Cyklů na 50MiB	Cyklů na 100MiB
1 měření	2 227 128 872	23 269 500 000	119 019 096 360	237 185 875 688
2 měření	2 285 063 688	23 681 680 876	118 809 922 286	237 550 006 474
3 měření	2 259 464 618	23 496 221 224	118 857 253 882	236 663 970 304
4 měření	2 271 139 036	23 505 594 960	118 751 787 182	237 706 264 478
5 měření	2 273 155 906	23 790 523 832	118 428 374 062	237 886 971 344
Průměr	2 263 190 424	23 548 704 179	118 773 286 755	237 398 617 658

Tab. 9.10: Měření rychlosti OFB šifrování pro 1MiB v závislosti na čase.

Velikost	Čas na 1MiB [µs]	Čas na 10MiB [µs]	Čas na 50MiB [µs]	Čas na 100MiB [µs]
Průměr	3 398 183,82	35 358 414,68	178 338 268,40	356 454 380,87
Průměr na 1MiB	3 398 183,82	3 535 841,47	3 566 765,37	3 564 543,81

Při porovnání tabulek 9.7 a 9.10 lze potvrdit, že šifrování pomocí OFB je lehce rychlejší než 32bitové CFB. Přesto lze říci, že šifrování celého souboru trvá déle než je očekáváno. Je to dáno použitou knihovnou FatFs a architekturou FAT systému, jelikož si knihovna alokuje pouze jeden sektor microSD karty, který má velikost 512 bytů. To způsobuje, že po každých 512 uložených bytech se musí alokovat další sektor a tím vzniká zpomalení při čtení a zápisu dat vždy, když se přechází na nový sektor. Průměrné čtení a zápis při změně sektoru jsou uvedeny v tabulce 9.11, kdy hodnoty pro měření byly náhodně vybrány z jednoho šifrování souboru. Jak lze vidět

v tabulce, průměrné hodnoty se sobě přibližně rovnají. Pro porovnání s rychlostí bez načítání sektoru slouží tabulka 9.12, která ukazuje průměrný čas pro čtení, zápis a šifrování. Hodnoty těchto časů byly zprůměrovány z několika stovek hodnot.

Tab. 9.11: Porovnání doby načítání sektoru provozních režimů CFB a OFB.

	CFB		OFB	
	Čas čtení [µs]	Čas zápisu [µs]	Čas čtení [µs]	Čas zápisu [µs]
Měření 1	312,92	1 183,53	314,77	1 027,05
Měření 2	314,11	1 106,42	314,78	1 122,95
Měření 3	313,96	1 107,59	314,64	1 337,26
Měření 4	314,16	1 105,19	314,57	1 100,05
Měření 5	314,15	1 202,34	314,65	1 105,12
Průměr	313,86	1 141,01	314,68	1 138,49

Budeme-li porovnávat časy v tabulce 9.12, musíme vzít v potaz, že CFB je 32bitové, takže šifrování bude pro stejnou velikost trvat dvojnásobný čas, než který je uveden v tabulce. Pro šifrování 8 bytů by byl výsledný čas pro CFB 5,48 µs a pro OFB 3,55 µs. Na šifrování 1MiB souboru je potřeba 131 072 opakování pro 8 bytů. Nepočítáme-li dobu potřebnou pro načtení sektoru, je zapotřebí 718 274,56 µs pro CFB a 465 305,6 µs pro OFB. Pokud navíc vypočítáme průměrný čas strávený pro alokaci sektoru, který probíhá každých 512 bytů, získáme 2 979 573,76 µs pro CFB a 2 976 071,68 µs pro OFB. Při sečtení časů pro alokaci sektoru a šifrování získáme podobný čas na 1MiB, jako jsou celkové změřené časy pro šifrování CFB a OFB. Tímto můžeme stanovit, že režie v podobě alokace sektorů je přibližně 80,56% pro CFB a 86,49% pro OFB. Propustnost šifrování bez režie by byla vypočtena dle vzorce $\frac{velikost\,[KiB]}{čas\,[s]}$. Pro CFB se dosadí hodnoty $\frac{1024}{0{,}71827456}$ a propustnost bez režií by byla rovna přibližně 1425,63 KiB/s. Stejným způsobem lze dosadit hodnoty pro OFB $\frac{1024}{0{,}4653056}$ a propustnost bez režie by byla rovna přibližně 2200,7 KiB/s.

Tab. 9.12: Časové porovnání fází šifrování provozních režimů CFB a OFB.

	CFB			OFB		
	Čtení [µs]	Zápis [µs]	Šifrování [µs]	Čtení [µs]	Zápis [µs]	Šifrování [µs]
Čas	0,82	0,84	1,08	1,39	0,87	1,29

9.3 Implementace šifrování pomocí UART rozhraní

Šifrování pomocí UART rozhraní je sekundární způsob šifrování, uživatel posílá data k šifrování a jsou mu vráceny šifrovaná data. Pro šifrování je využita 8bitová CFB, kdy klíč a inicializační vektor jsou generovány pomocí LFSR a na konci šifrování jsou předávány uživateli.

Proces šifrování je jednoduší a méně náročný na straně šifrátoru než v případě microSD karty, ale pro uživatele více náročnější jelikož musí řešit rychlost odesílání dat na své straně. Šifrování je vázáno na tlačítko BTN0, které při stisku spustí šifrování, jeho následné stisknutí ukončí šifrování a odešle se klíč a IV. Šifrování probíhá ve třech 4 fázích, kdy druhá až čtvrtá fáze probíhá v cyklu:

1. Generace klíče a IV.
2. Příjem bytu od uživatele.
3. Šifrování bytu.
4. Odeslání bytu.

V první fázi se vygeneruje klíč a IV. Klíč je uložen do pole 8bitových celých čísel bez znaménka o velikosti 10 a načten do IP bloku šifry LBlock. IV je uložen jako dvě 32bitová celá čísla bez znaménka pro potřeby šifrování a pro uchování počátečního IV je použito osm polí 8bitových celých čísel bez znaménka.

V druhé fázi se čeká na příjem dat od uživatele, kdy velikost příjmu je vždy jeden byte. Pokud uživatel posílá více bytů, tak UART využívá FIFO (First In, First Out) „buffer“ neboli frontu. Fronta má omezenou velikost, takže si uživatel musí hlídat, aby nebyla přeplněna, což by vyústilo ve ztrátu některých bytů.

Ve třetí fázi je uskutečněno samotné šifrování jednoho bytu pomocí 8bitového CFB. Šifrování probíhá získáním šifrovaného IV do dvou 32 bitových celých čísel bez znaménka. U prvního vektoru se provede bitová rotace doleva o 8 bitů. Poté je na tento vektor aplikována operace XOR společně s přijatým bytem. Poslední byte XORovaného vektoru se uloží do výstupní proměnné a ve vektoru ho nahradí nejvíce signifikantní byte z druhého vektoru. Na druhý vektor je následně použita bitová rotace doleva o 8 bitů a její poslední byte je nahrazen hodnotou ve výstupní proměnné. Tyto dva vektory jsou poté použity jako nové vstupní vektory pro šifrování.

Ve čtvrté fázi dochází k odeslání hodnoty výstupní proměnné z třetí fáze. Pokud bylo stisknuto tlačítko pro ukončení šifrování, pak v této fázi dochází také k odeslání klíče a IV. Po úspěšném odeslání klíče a IV je dioda rozsvícena zeleně.

Podobně jako v případě šifrování na microSD kartu proběhlo měření rychlosti šifrování. Pro samostatné šifrování jednoho znaku je potřeba průměrně 0,96 µs. Dobu potřebnou k provedení celého šifrování nelze vypočítat, jelikož je ovlivněna lidským faktorem, kdy od zadání vstupu do doby stisknutí tlačítka pro ukončení nelze stanovit přibližný čas, který by neovlivnil rychlost šifrování. Lze tedy pouze vypočítat, že

čas potřebný pro čisté šifrování 1MiB je přibližně 1 006 632,96 µs, což je 1,01 sekundy.

Z tohoto odvozeného času lze spočítat, jaká by byla přibližná propustnost šifry. Výpočet lze provést dosazením do vzorce $\frac{velikost\,[KiB]}{čas\,[s]}$. Po dosazení hodnot $\frac{1024}{1{,}00663296}$ vyjde přibližná propustnost 1017,25 KiB/s.

10 Porovnání s dalšími prácemi

V poslední kapitole se porovnává tato implementace šifrátoru s dalšími implementacemi známých šifer. Jelikož se nepodařilo dohledat implementace, které by využívaly procesní systém a programovací logiku společně, k porovnání jsou použity pouze implementace na programovací logice. Implementace se porovnává jak s jinými implementacemi šifry LBlock tak i s implementacemi šifer PRESENT a AES. Hodnoty pro porovnání byly převzaty z článků Wenling, W. (2011, [30]) Aljazeera, K. R. (2016, [36]), Moraitis, S. (2020, [37]) pro šifru LBlock, z článku Gomez, E. (2017, [38]) pro šifru PRESENT a z článku Hämäläinen, P. (2006, [39]) pro šifru AES.

Tabulka 10.1 porovnává vlastní implementaci a jednotlivé implementace z článků zmíněných výše. Nutno říci, že většina implementací cílila na co největší propustnost, zatímco tato implementace má navíc za úkol komunikaci mezi procesním systémem a programovací logikou, při které vzniká režie.

Jediná implementace, která se zaměřila na co nejmenší hardwarové nároky je AES [39] a cílila na 8bitový ASIC.

Hodnoty implementace LBlock [30] jsou z originálního návrhového článku, kde byly změřeny na výrazně omezeném hardwaru.

LBlock [36] se zaměřil na co největší propustnost dat, kdy hardwarové nároky byly více jak trojnásobné při porovnání s implementací popsanou v této práci. LBlock [36] byl implementován v podobě IP bloku, u kterého není z článku zřejmé, jestli jsou data a klíč děleny na menší bloky, které jsou načítány postupně nebo jsou načítány zároveň.

LBlock [37] využil „unrolled“ architektury využívající 2 moduly, které paralelně vypočítávají data pro kolo a klíč. Využitá logická plocha je v článku měřena v řezech, kterých je přibližně o 1,5 krát více než v případě implementace popsané v této práci.

PRESENT [38] je podobně jako předcházející šifry také zaměřen na výkon s co nejmenším využitím hardwarových prostředků. Šifra využívá 193 řezů, to je přibližně 46% v porovnání s implementací v této práci, musí se však vzít v potaz, že šifra PRESENT nevyužívá Feistelovu síť.

Jak už bylo zmíněno výše, nelze tuto implementaci porovnat s implementacemi v článcích, protože cílem práce nebylo měřit režii při komunikaci mezi procesním systémem a programovací logikou a ani měření propustnosti vytvořeného IP bloku na programovací logice. Důvodem velkého rozdílu v propustnosti může být, že dané šifry načítají klíč i data v jednom kole, zatímco implementace v této práci načítá data a klíč po 32bitových blocích v 5 kolech.

Tab. 10.1: Porovnání implementací [30][36][37][38][39].

Šifra	Propustnost [MB/s]	Frekvence [MHz]	Zařízení
Vlastní implementace microSD CFB reálná	0.28	125	Zynq-7000 + Zybo-Z7
Vlastní implementace microSD CFB teoretická	1,46	125	Zynq-7000 + Zybo-Z7
Vlastní implementace microSD OFB reálná	0,3	125	Zynq-7000 + Zybo-Z7
Vlastní implementace microSD OFB teoretická	2,25	125	Zynq-7000 + Zybo-Z7
Vlastní implementace UART teoretická	1,04	125	Zynq-7000 + Zybo-Z7
LBlock [30]	0,025	0,1	0.18µm CMOS technologie
LBlock [36]	115,54	120,174	Spartan-6
LBlock [37]	80,88	323	Kintex-7
PRESENT [38]	36,4	145,6	Spartan 3AN
AES [39]	15,13	152	8bitový ASIC

Závěr

Cílem bakalářské práce bylo seznámení se šiframi lehké kryptografie a vytvoření šifrátoru, který implementuje jednu vybranou šifru v jazyce VHDL. Šifra byla nejprve volena tak, aby splňovala požadavek na šifrování souborů. Z důvodu uvedeného požadavku byly zvoleny blokové šifry, které jsou pro toto použití nejvhodnější a na rozdíl od hašovacích funkcí je lze dešifrovat. Výběr probíhal mezi šiframi LBlock, LED, mCRYPTON, PRESENT a SIMON. Jako nejvhodnější šifra byla zvolena šifra LBlock, druhou v pořadí je šifra PRESENT.

Zvolená šifra LBlock byla poté implementována v několika modulech. Řídící modul obsahuje sekvenční logiku a pomocí něj se šifrují data. Pro každý modul byly vytvořeny testy, kdy funkce každého modulu byla úspěšně ověřena. Pro hlavní modul byly použity vstupní hodnoty z oficiální dokumentace šifry LBlock. Tato šifra byla poté integrována do vlastního IP bloku. Dále byl vytvořen modul ke generaci 32bitových pseudonáhodných čísel.

Pro implementaci na vývojovou desku ZYBO Z7 byl navržen blokový návrh sloužící pro propojení programovací logiky a procesního systému FPGA čipu řady Zynq-7000. Blokový návrh obsahuje jak prvky vývojové desky, tak i interní prvky pro FPGA čip. Prvky vývojové desky jsou RGB LED a tlačítka. Interními prvky jsou pomocné bloky, procesní systém a vytvořený IP blok. Pro blokový návrh a jeho prvky byly zjištěny hardwarové nároky, které byly vyhodnoceny jako minimální, jelikož využívají přibližně 2% dostupných hardwarových zdrojů.

Práce se vstupními soubory jsou zajišťovány na procesním systému, kdy programovatelná logika zajišťuje urychlování šifrování. Pro šifrování souborů je možné využít microSD karty, navržené jako primární šifrování nebo UART rozhraní, navržené jako sekundární. Pomocí microSD karty lze šifrovat třemi způsoby, kterými jsou 32bitové CFB s možností generace klíče, 32bitové CFB s možností použití vlastního klíče nebo OFB s použitím vlastního klíče. Pomocí UART lze přenášet data, která jsou šifrována pomocí 8bitového CFB.

Rychlost šifrování pro microSD kartu byla změřena pro CFB a OFB bez generace klíče. Výsledné hodnoty měření byly vyšší než očekávané. Důvodem je alokace nového sektoru pro čtení a zápis. Výsledky měření samotného šifrování na 1MiB byly 0,72 s pro CFB a 0,67 s pro OFB. Propustnost pro CFB je 271,73 KiB/s a pro OFB je 291,21 KiB/s. Měření na rozhraní UART probíhalo pouze pro šifrování bez přijímání a odesílání dat. Jelikož se jednalo o 8bitové CFB, bylo šifrování mírně pomalejší, šifrovat 1MiB by trvalo 1,01 s. Teoretická propustnost pro UART je 1017,25 KiB/s.

Možností rozšíření práce je zaměřit se na optimalizaci a zrychlení šifrování na microSD kartu. Druhým možným rozšířením by byla implementace šifry PRESENT a porovnání její náročnosti a rychlosti šifrování s implementovanou šifrou LBlock.

Literatura

[1] SEKANINA, Lukáš. *Evolvable components: from theory to hardware implementations.* Berlin: Springer-Verlag, 2004. ISBN 3-540-40377-9.

[2] CHANDRASEKHAR, Vikram. *CAD for a 3-dimensional FPGA* [online]. 2007 [cit. 16. 10. 2020]. Dostupné z URL: < `http://dspace.mit.edu/handle/1721.1/40520`> Diplomová práce. Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science.

[3] Introduction to FPGA Structure, Components, Applications, *Electronicshub.org* [online]. 2020 [cit. 16. 10. 2020]. Dostupné z URL: <`https://www.electronicshub.org/wp-content/uploads/2020/01/FPGA-Structure.jpg`>

[4] ZEIDMAN, Bob. *All about FPGAs.* EE Times [online]. 2006 [cit. 15. 5. 2021]. Dostupné z URL: <`https://www.eetimes.com/all-about-fpgas/`>

[5] COFER, R.C. a Benjamin F. HARDING. Rapid System Prototyping with FPGAs. *Encyclopedia of Cryptography and Security* [online]. Boston, MA: Springer US, 2011, 2006 [cit. 15. 5. 2021]. ISBN 978-1-4419-5905-8. Dostupné z: doi: <`https://doi.org/10.1016/B978-0-7506-7866-7.X5000-8`>

[6] XILINX, INC. *Zynq-7000 SoC Data Sheet: Overview* [online]. DS190 (v1.11.1) July 2, 2018. [cit. 17. 10. 2020]. Dostupné z URL: <`https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf`>

[7] Digilentinc.com *ZYBO Z7. Reference manual.* [online]. Digilent: © 2020. [cit. 16. 4. 2021]. Dostupné z URL: <`https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual`>

[8] BURDA, Karel. *Aplikovaná kryptografie.* Brno: VUTIUm, 2013. ISBN 978-80-214-4612-0.

[9] BURDA, Karel. *Úvod do kryptografie.* Brno: Akademické nakladatelství CERM, 2015. ISBN 978-80-7204-925-7.

[10] MAO, Wenbo. *Modern Cryptography: Theory and Practice.* 5th ed. Upper Saddle River: Prentice Hall, 2004. ISBN 0-13-066943-1.

[11] SMART, Nigel Paul. *Cryptography: An Introduction.* London: McGraw-Hill College, 2003. ISBN 0-077-09987-7.

[12] HAVLÍČEK, Jiří. *Šifrovací algoritmy lehké kryptografie* [online]. Brno, 2013 [cit. 22. 10. 2020]. Dostupné z URL: <https://www.vutbr.cz/studenti/zav-prace/detail/66565> Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Ing. Vlastimil Člupek.

[13] FONTAINE, Caroline. Synchronous Stream Cipher. *Encyclopedia of Cryptography and Security* [online]. Boston, MA: Springer US, 2011 [cit. 22. 10. 2020]. ISBN 978-1-4419-5905-8. Dostupné z: doi: <https://doi.org/10.1007/978-1-4419-5906-5_376>

[14] FONTAINE, Caroline. Self-Synchronizing Stream Cipher. *Encyclopedia of Cryptography and Security* [online]. Boston, MA: Springer US, 2011 [cit. 22. 10. 2020]. ISBN 978-1-4419-5905-8. Dostupné z: doi: <https://doi.org/10.1007/978-1-4419-5906-5_371>

[15] Feistel cipher. *Wikipedia* [online]. [cit. 3. 11. 2020]. Dostupné z URL: <https://en.wikipedia.org/wiki/Feistel_cipher#/media/File:Feistel_cipher_diagram_en.svg>

[16] PRENEEL, Bart. Modes of Operation of a Block Cipher *Encyclopedia of Cryptography and Security* [online]. Springer, Boston, MA, 2011 [cit. 17. 4. 2021]. ISBN 978-1-4419-5906-5. Dostupné z: doi: <https://doi.org/10.1007/978-1-4419-5906-5_599>

[17] D.I. Management Services Pty Ltd. Cryptosys.net [online]. ©2004-21 [cit. 17. 4. 2021]. Dostupné z URL: <https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html>

[18] DWORKIN, Morris. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *National Institute of Standards and Technology Special Publication 800-38D* [online]. 2007, 37 s. [cit. 19. 5. 2021]. Dostupné z URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>

[19] Cipher block chaining (CBC). *Wikipedia* [online]. [cit. 17. 4. 2021]. Dostupné z URL: <https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg>

[20] DWORKIN, Morris. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *National Institute of Standards and Technology Special Publication 800-38A* [online]. Washington: U.S. government printing office Washington 2001, 66 s. [cit. 17. 4. 2021]. Dostupné z URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>

[21] Counter (CTR). *Wikipedia* [online]. [cit. 17. 4. 2021]. Dostupné z URL: <https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CTR_encryption_2.svg>

[22] ŠALDA, Jakub. *Lehká kryptografie* [online]. Praha, 2017 [cit. 6. 11. 2020]. Dostupné z URL: <https://vskp.vse.cz/69739_lehka_kryptografie.> Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce RNDr. Radomír Palovský, CSc.

[23] POSCHMANN, Axel. *LIGHTWEIGHT CRYPTOGRAPHY: Cryptographic Engineering for a Pervasive World* [online]. Bochum, 2009 [cit. 3. 11. 2020]. Dostupné z URL: <https://www.ei.ruhr-uni-bochum.de/media/crypto/veroeffentlichungen/2011/09/16/thesisp.pdf> Disertační práce. Ruhr-University Bochum, Germany, Faculty of Electrical Engineering and Information Technology.

[24] YALLA, Panasayya a Jens-Peter KAPS. Lightweight Cryptography for FPGAs: An Ultra-Lightweight Block Cipher. *2009 International Conference on Reconfigurable Computing and FPGAs* [online]. Berlin, Heidelberg: IEEE, 2009 s. 225-230 [cit. 5. 3. 2021]. ISBN 978-1-4244-5293-4. Dostupné z: doi: <https://doi.org/10.1109/ReConFig.2009.54>

[25] JEDLIČKA, Jakub. CUSTOM INTELLECTUAL PROPERTY BLOCK FOR LBLOCKCIPHER. In: *Proceedings I of the 27th Conference STUDENT EEICT 2021.* Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2021, ISBN 978-80-214-5942-7. Dostupné také z URL: <https://conf.feec.vutbr.cz/eeict/index/pages/view/ke_stazeni>. Přijato k publikování rozsah stran 4.

[26] KLÍMA, Vlastimil. *Co je to lehká kryptografie?* [online]. 2011 [cit. 3. 11. 2020]. Dostupné z URL: <https://cryptography.hyperlink.cz/2011/ST_2011_10_16_17.pdf>

[27] HAMMAD, B.T., N. JAMIL, M.E. RUSLI a M.R. Z'ABA. A survey of Lightweight Cryptographic Hash Function. *International Journal of Scientific & Engineering Research* [online]. 2017, **2017(8)** [cit. 5. 11. 2020]. ISSN

2229-5518. Dostupné z URL: <https://www.ijser.org/researchpaper/A-survey-of-Lightweight-Cryptographic-Hash-Function.pdf>

[28] NEKUŽA, Karel. *Odlehčená kryptografie pro embedded zařízení* [online]. Brno, 2016 [cit. 6. 11. 2020]. Dostupné z URL: <https://www.vutbr.cz/studenti/zav-prace?zp_id=93665.> Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Ing. Zdeněk Martinásek, Ph.D.

[29] BOGDANOV Andrey, L. R. KNUDSEN, G. LEANDER, Christof PAAR, Axel POSCHMANN, M.J.B. ROBSHAW, Y. SEURIN a C. VIKKELSOE. PRESENT: An Ultra-Lightweight Block Cipher. *Cryptographic Hardware and Embedded Systems - CHES 2007.* [online]. Berlin: Springer, Berlin, Heidelberg, 2007, s. 455-466 [cit. 7. 11. 2020]. ISBN 978-3-540-74735-2. Dostupné z: doi: <https://doi.org/10.1007/978-3-540-74735-2_31>

[30] WENLING, Wu a Lei ZHANG. LBlock: A Lightweight Block Cipher. *Applied Cryptography and Network Security* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 327-344 [cit. 8. 11. 2020]. ISBN 978-3-642-21554-4. Dostupné z: doi: <https://doi.org/10.1007/978-3-642-21554-4_19>

[31] MISHRA, Shivshankar, Ram Racksha TRIPATHI a Devendra Kr. TRIPATHI. Implementation of configurable linear feedback shift register in VHDL: A Lightweight Block Cipher. *Applied Cryptography and Network Security* [online]. Berlin, Heidelberg: IEEE, 2016, 2016, s. 342-346 [cit. 5. 4. 2021]. ISBN 978-1-5090-2118-5. Dostupné z: doi: <https://doi.org/10.1109/ICETEESES.2016.7581406>

[32] ALFKE, Peter. *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators: XAPP 052* [online]. 7.7.1996 [cit. 5. 4. 2021]. Dostupné z URL: <https://www.xilinx.com/support/documentation/application_notes/xapp052.pdf>

[33] ARM LIMITED. *AMBA® AXI and ACE: Protocol Specification.* H.c verze 2.2. Cambridge: Arm Limited, 2021. [cit. 6. 4. 2021]. Dostupné z URL: <https://developer.arm.com/documentation/ihi0022/latest>

[34] XILINX. *AXI Reference Guide* [online]. UG761 (v14.3). 2012. Dostupné z URL: <https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf>

[35] DIEHL, William, Farnoud FARAHMAND, Panasayya YALLA, Jens-Peter KAPS a Kris GAJ. Comparison of hardware and software implementations

of selected lightweight block ciphers. *2017 27th International Conference on Field Programmable Logic and Applications (FPL)* [online]. Berlin, Heidelberg: IEEE, 2017, 2017, s. 1-4 [cit. 23. 3. 2021]. ISBN 978-9-0903-0428-1. Dostupné z: doi: <https://doi.org/10.23919/FPL.2017.8056808>

[36] ALJAZEERA, K. R., R. NANDAKUMAR a S. B. ERSHAD. Design and characterization of LBlock cryptocore. *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)* [online]. IEEE, 2016, s. 166-172 [cit. 20. 5. 2021]. ISBN 978-1-5090-4620-1. Dostupné z: doi: <https://doi.org/10.1109/SCOPES.2016.7955732>

[37] MORAITIS, S., D. SEITANIDIS, G. THEODORIDIS a O. KOUFOPAVLOU. Exploring the FPGA Implementations of the LBlock, Piccolo, Twine, and Klein Ciphers. *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)* [online]. IEEE, 2020, s. 46-51 [cit. 20. 5. 2021]. Dostupné z: doi: <https://doi.org/10.1109/VLSI-SOC46417.2020.9344108>

[38] GOMEZ, Edwar, Cesar HERNANDEZ a Fredy MARTINEZ. Performance evaluation of the present cryptographic algorithm over FPGA. *Contemporary Engineering Sciences* [online]. 2017, **10**, s. 555-567 [cit. 20. 5. 2021]. ISSN 13147641. Dostupné z: doi: <https://doi.org/10.12988/ces.2017.7653>

[39] HÄMÄLÄINEN, P., T. ALHO, M. HANNIKAINEN a T.D. HAMALAINEN. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. *9th EUROMICRO Conference on Digital System Design (DSD'06)* [online]. IEEE, 2006, s. 577-583 [cit. 20. 5. 2021]. ISBN 0-7695-2609-8. Dostupné z: doi: <https://doi.org/10.1109/DSD.2006.40>

Seznam symbolů a zkratek

AES Standard pokročilého šifrování – Advanced Encryption Standard

AMBA Pokročilá architektura mikrokontrolérové sběrnice – Advanced Microcontroller Bus Architecture

ANSI Americký národní normalizační institut – American National Standards Institute

ARM Pokročilý procesor s redukovanou instrukční sadou – Advanced Reduced Instruction Set Computer Machine

ASCII Americký standardní kód pro výměnu informací – American Standard Code for Information Interchange

ASIC Integrované obvody určené pro aplikaci – Application Specific Integrated Circuit

AXI Pokročil rozšiřitelné rozhraní – Advanced eXtensible Interface

BIBUF Obousměrná vyrovnávací paměť – Bi-Directional Buffer

CBC Řetězení šifrových bloků – Cipher Block Chaining

CFC Šifrová zpětná vazba – Cipher Feedback

CLB Konfigurovatelné logické bloky – Configurable Logic Blocks

CTR Čítačový režim – Counter

CPU Centrální procesorová jednotka – Central Processing Unit

DDR3 Double Data Rate verze 3 – Double Data Rate 3

DDR3L Double Data Rate verze 3 s nízkým napětím – Double Data Rate 3 Low voltage

DES Standart datového šifrování – Data Encryption Standard

ECB Kódová kniha – Electronic Codebook

FAT Alokační tabulka souborů – File Allocation Table

FF Bistabilní klopný obvod – Flip-Flop

FIFO První dovnitř, první ven (fronta) – First In, First Out (queue)

FDCE Bistabilní klopný obvod typu D s aktivními hodinami a asynchronním signálem clear – D Flip-Flop with Clock Enable and Asynchronous Clear

FDPE Bistabilní klopný obvod typu D s aktivními hodinami a asynchronním signálem preset – Flip-Flop with Clock Enable and Asynchronous Preset

FDRE Bistabilní klopný obvod typu D s aktivními hodinami a synchronním signálem reset – D Flip-Flop with Clock Enable and Synchronous Reset

FDSE Bistabilní klopný obvod typu D s aktivními hodinami a synchronním signálem set – D Flip-Flop with Clock Enable and Synchronous Set

FPGA Programovatelná hradlová pole – Field-Programmable Gate Array

GCM Galoisovský/čítačový režim – Galois/Counter Mode

GE Ekvivalent brány – Gate Equivalent

GPIO Univerzální vstupní/výstupní pin – General-purpose input/output

GSM Globální systém pro mobilní komunikaci – Global System for Mobile Communications

HDMI Multimediální rozhraní s vysokým rozlišením – High-Definition Multimedia Interface

I/O Vstup/Výstup – Input/Output

IBUF Vstupní vyrovnávací paměť – Input Buffer

IC Integrovaný obvod – Integrated Circuit

IDE Vývojové prostředí – Integrated Development Environment

IEC Mezinárodní elektrotechnická komise – International Electrotechnical Commission

IOB Vstupně/výstupní blok – Input/Output Block

ISO Mezinárodní organizace pro normalizaci – International Organization for Standardization

IV Inicializační vektor – Initialization Vector

IP blok Intelectual property blok – Intellectual Property block

LED Elektroluminiscenční dioda – Light-Emitting Diode

LFSR Posuvný registr s lineární zpětnou vazbou – Linear-Feedback Shift Register

LUT Vyhledávací tabulka – Look-Up Table

microSD Miniaturizovaná Secure Digital – miniaturized Secure Digital

MUX Multiplexor – Multiplexer

MUXF7 Multiplexor s vyhledávací tabulkou 2:1 s obecným výstupem – 2-to-1 Look-Up Table Multiplexer with General Output

NAND Negovaný logický součin – Negation of Logical Conjunction

OBUF Výstupní vyrovnávací paměť – Output Buffer

OFB Výstupní zpětná vazba – Output Feedback

P-box Permutační tabulka – Permutation box

PIB Programovatelný propojovací blok – Proglammable Interconnect Blocks

PKCS Kryptografický standard veřejného klíče – Public-Key Cryptography Standards

PL Programovací logika – Programmable Logic

PS Procesní systém – Processing System

RFID Identifikace na rádiové frekvenci – Radio Frequency Identification

RJ45 Registrovaná koncovka 45 – Registered Jack 45

RGB Červená - zelená - modrá – Red - Green - Blue

RSA Rivest - Shamir - Adleman – Rivest - Shamir - Adleman

RTL Úroveň meziregistrových přenosů – Register Transfer Level

S-box Zaměňovací tabulka – Substitution box

SHA Bezpečný hašovací algoritmus – Secure Hash Algorithm

SPN Substitučně-permutační síť – Substitution-Permutation Network

SRAM Statická paměť s přímým přístupem – Static Random Access Memory

UART Univerzální asynchronní přijímač – vysílač – Universal Asynchronous Receiver-Transmitter

USB Univerzální sériová sběrnice – Universal Serial Bus

VHDL Hardwarový deskriptivní jazyk pro velmi rychlé integrované obvody – Very High Speed Integrated Circuit Hardware Description Language

XNOR Negovaná exkluzivní disjunkce – Negated Exclusive Disjunction

XOR Exkluzivní disjunkce – Exclusive Disjunction

A Obsah přílohy

Kapitola obsahuje popis příloh. Přílohy zahrnují z velké části kód napsaný ve VHDL a C. Složka *PL_sources* obsahuje zdrojové soubory pro blokový návrh ve složce *design*, pro IP blok ve složce *IP_block*, pro posuvný registr s lineární zpětnou vazbou ve složce *LFSR* a pro testy jednotlivých modulů ve složce *simulations*. Ve složce *IP_block* se nachází důležité soubory pro IP blok a složka *sources_LBlock*, která obsahuje zdrojové soubory implementace šifry LBlock.

Složka *PS_sources* obsahuje vyexportovaný blokový návrh obsahující bitstream a sloužící k propojení s PL. Dále se zde nachází složka *logic*, ve které se nachází zdrojové soubory v jazyce C a jejich hlavičkové soubory využité pro šifrátor.

Kořenový adresář dále obsahuje soubor *BOOT.bin*, který slouží pro bootovaní šifrátoru ve formě už vyrobené aplikace a soubor README.md obsahují tento popis příloh a stručný návod k práci se soubory.

Všechny soubory lze také nalézt v online repozitáři na Githubu s url adresou `https://github.com/jedla97/HW-encoder-for-lightweigth-cryptography`.

- `/` ..... kořenový adresář
  - `PL_sources` ..... zdrojové soubory pro programovací logiku
    - `design` ..... zdrojové soubory pro blokový návrh
    - `IP_block` ..... zdrojové soubory pro IP blok
      - `sources_LBlock` ..... zdrojové soubory pro šifru LBlock
      - `component.xml`
      - `LBlock_wrapper_v1_0.vhd`
      - `LBlock_wrapper_v1_0_S00_AXI.vhd`
      - `LBlock_wrapper_v4_0.tcl`
    - `LFSR` ..... zdrojový soubor pro LFSR
    - `simulations` ..... zdrojové soubory pro testování modulů šifry LBlock
  - `PS_sources` ..... zdrojové soubory pro procesní systém
    - `logic` ..... zdrojový soubor pro šifrátor
    - `LBlock_design.xsa` ..... vyexportovaný blokový návrh
  - `BOOT.bin` ..... soubor pro bootování šifrátoru
  - `README.md` ..... popis příloh a stručný návod